

EuroBachTeX 2007

TUGboat, Volume 29, Number 1, 2008

Buletyn GUST, Zeszyt 24, 2007

Die TeXnische Komödie, 20. Jahrgang, 1/2008



EuroBachTeX 2007	2	Conference program, delegates, and sponsors
	6	Jerzy Ludwiczowski and Petr Sojka / <i>EuroBachTeX 2007: Paths to the Future</i>
	13	Sam Guravage / <i>Confessions of a teenage TeX user</i>
Typography	14	Grażyna Jackowska / <i>Handmade paper: A mixture of handcraft, art and fun</i>
	16	Andrzej Tomaszewski / <i>Designing a special book: With both pleasure and ... fear</i>
	20	Dorota Cendrowska / <i>Enumerations as an interesting form of text appearance</i>
Fonts	25	Jerzy Ludwiczowski, Bogusław Jackowski and Janusz Nowacki / <i>Five years after: Report on international TeX font projects</i>
	27	Janusz Nowacki / <i>Cyklop: A new font family</i>
	28	Hans Hagen / <i>Do we need a font system in TeX?</i>
	34	Taco Hoekwater / <i>OpenType fonts in LuaTeX</i>
	36	Hàn Thế Thành / <i>Font-specific issues in pdfTeX</i>
	42	Karel Horák / <i>Those obscure accents ...</i>
	45	Klaus Höppner / <i>Creation of a PostScript Type 1 logo font with MetaType 1</i>
	50	Karel Píška / <i>Procedures for font comparison</i>
	57	Karel Píška / <i>Comments and suggestions about the Latin Modern fonts</i>
	66	Jerzy Ludwiczowski and Karl Berry / <i>The GUST Font License: An application of the L^AT_EX Project Public License</i>
Resources	68	Arthur Reutenauer / <i>A brief history of TeX, volume II</i>
	73	Ulrik Vieth / <i>Overview of the TeX historic archive</i>
	77	Joanna Ludmiła Ryćko / <i>TeX Clinic</i>
Multilingual Document Processing	79	Ameer Sherif and Hossam Fahmy / <i>Parameterized Arabic font development for AlQalam</i>
	89	Atif Gulzar and Shafiq ur Rahman / <i>Nastaleeq: A challenge accepted by Omega</i>
	95	Hàn Thế Thành / <i>Typesetting Vietnamese with VnTeX (and with the TeX Gyre fonts too)</i>
	101	Jean-Michel Hufflen / <i>Managing order relations in MIBIBTeX</i>
Electronic Documents	109	Jean-Michel Hufflen / <i>Introducing L^AT_EX users to XSL-FO</i>
	117	Tomasz Łuczak / <i>Using TeX in a wiki</i>
Publishing	118	Petr Sojka and Michal Růžička / <i>Single-source publishing in multiple formats for different output devices</i>
	125	Péter Szabó / <i>Practical journal and proceedings publication on paper and on the web</i>
Software & Tools	133	Jim Hefferon / <i>An experimental CTAN upload process</i>
	136	Norbert Preining / <i>TeX (Live) on Debian</i>
	140	Siep Kroonenberg / <i>Epspdf: Easy conversion between PostScript and PDF</i>
	143	Martin Schröder / <i>pdfTeX 1.40: What's new</i>
	146	Jonathan Kew / <i>X_YTeX Live</i>
	151	Gerd Neugebauer / <i>Conventional scoping of registers — An experiment in ε_XTeX</i>
	157	Jean-Michel Hufflen / <i>MIBIBTeX: Reporting the experience</i>
	163	David Kastrup / <i>Writing (L^A)TeX documents with AUCTeX in Emacs</i>
	164	Tomasz Łuczak / <i>LyX: An editor not just for secretaries</i>
	166	Péter Szabó / <i>Automated DVD menu authoring with pdfL^AT_EX</i>
L^AT_EX	176	Zofia Walczak / <i>Graphics in L^AT_EX using TikZ</i>
	180	Grzegorz Murzynowski / <i>L^AT_EX vs. L^AT_EX — a modification of the logo</i>
	181	David Kastrup / <i>Benefits, care and feeding of the bigfoot package</i>
	184	Johannes Große / <i>MathPSfrag: L^AT_EX labels in Mathematica plots</i>
	190	David Kastrup / <i>makematch, a L^AT_EX package for pattern matching with wildcards</i>
	193	David Kastrup / <i>qstest, a L^AT_EX package for unit tests</i>
Macros	199	Grzegorz Murzynowski / <i>gmverse and gmcontinuo — some nontrivial placement of text on a page</i>
	201	Grzegorz Murzynowski / <i>The gmdoc bundle — a new tool for documenting (L^A)TeX sources</i>
Hints & Tricks	207	Paweł Jackowski / <i>TeX beauties and oddities: A permanent call for TeX pearls</i>
Puzzle	216	Janusz Nowacki / <i>Crossword</i>
Abstracts	217	Abstracts (Hagen, Hoekwater, Kastrup, Lotz, Moore, Ryćko, Tomaszewski)
ConTeXt	219	Aditya Mahajan / <i>ConTeXt basics for users: Table macros II</i>
TUG Business	223	TUG institutional members
Advertisements	223	TeX consulting and production services
Memorial	224	Maurice Laugier / <i>In memoriam Bernard Gaulle</i>

Copyright © 2007 T_EX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the T_EX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

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

Biuletyn Polskiej Grupy Użytkowników Systemu T_EX (ISSN 1230-5630) is published by GUST.

Die T_EXnische Komödie (ISSN 1434-5897) is published by DANTE e.V.

This publication was distributed by the Czechoslovak T_EX Users Group to its members.

For web sites and information on all the T_EX user groups, see <http://tug.org/usergroups.html>. Please join the user group best for you!

The script typeface used on the covers and title page of this issue is Vassallo 1.0, created by Jefferson J. Vorzimmer, «bang the drum» graphics.

T_EX is a trademark of the American Mathematical Society.

Printed in Germany, December 2007.



EuroBachTeX 2007

Paths to the Future

TUGboat Volume 29, Number 1, 2008

Biuletyn GUST, Zeszyt 24, 2007

Die TEXnische Komödie, 20. Jahrgang, 1/2008



Bachotek 28.04-2.05.2007

EuroBachTeX 2007

Paths to the Future

XVII European TeX Conference

Bachotek, Poland

April 28 - May 2, 2007

TUGboat Volume 29, Number 1, 2008

Buletyn GUST, Zeszyt 24, 2007

Die TeXnische Komödie, 20. Jahrgang, 1/2008

PROCEEDINGS EDITORS

Barbara Beeton

Karl Berry

Jerzy Ludwichowski

Tomasz Przechlewski

Stanisław Wawrykiewicz



EuroBachTeX 2007 — Paths to the future

XVII European TeX Conference • XV GUST conference

April 28–May 2, 2007 • Bachotek, Poland



Organizers

Československé sdružení uživatelů TeXu (CSTUG) • Polska Grupa Użytkowników systemu TeX (GUST)

Organizing committee

Jerzy Ludwichowski (Chairman) • Petr Sojka (Co-chairman)
• Bogusław Jackowski • Taco Hoekwater • Volker RW Schaa • Jolanta Szelatyńska

Program committee

Bogusław Jackowski (Chairman) • Hans Hagen (Co-chairman)
• Petr Sojka • Jonathan Kew • Jerzy Ludwichowski

Sponsors

Polish Ministry of Science and Higher Education, <http://www.nauka.gov.pl>
Information & Communication Technology Centre, Nicolaus Copernicus University, <http://www.uci.umk.pl>
Institute of Physics, Nicolaus Copernicus University, <http://www.fizyka.umk.pl/phys>
Focal Image Ltd, <http://www.focalimage.com>
Sun Microsystems Poland Sp. z o.o., <http://pl.sun.com>
TeX Users Group (TUG), <http://www.tug.org>
Die Deutschsprachige Anwendervereinigung TeX e.V. (DANTE), <http://www.dante.de>
De Nederlandstalige TeX Gebruikersgroep (NTG), <http://www.ntg.nl>
Československé sdružení uživatelů TeXu (CSTUG), <http://www.cstug.cz>
Polska Grupa Użytkowników Systemu TeX (GUST), <http://www.gust.org.pl>

Participants

<i>Alexander S. Berdnikov</i> , Institute of Analytical Instrumentation, Russian Academy of Sciences, Russian Federation	<i>Janusz Gumkowski</i> , Uniwersytet Mikołaja Kopernika, Uczelniane Centrum Informatyczne, Poland
<i>Piotr Bolek</i> , 7bull.com Sp. z o.o., Poland	<i>Michael A. Guravage</i> , Literate Solutions, The Netherlands
<i>Maria Bolek</i> , Poland	<i>Sam Guravage</i> , The Netherlands
<i>Danuta Bolek</i> , Poland	<i>Hans Hagen</i> , PRAGMA ADE, The Netherlands
<i>Andrzej Borzyszkowski</i> , GUST, Poland	<i>Taco Hoekwater</i> , Elvenkind BV, The Netherlands
<i>Jarosław Brykalski</i> , Poland	<i>Klaus Höppner</i> , DANTE e.V., Germany
<i>Dorota Cendrowska</i> , Poland	<i>Karel Horák</i> , Mathematical Institute, Academy of Science, Czech Republic
<i>Natalia Chlebus</i> , Uniwersytet Gdański, Instytut Oceanografii, Poland	<i>Jean-Michel Hufflen</i> , LIFC — University of Franche-Comté, France
<i>Leszek Czerwiński</i> , Poland	<i>Marise Hufflen</i> , France
<i>Marek Czubenko</i> , Uniwersytet Mikołaja Kopernika, Uczelniane Centrum Informatyczne, Poland	<i>Andrzej Icha</i> , Pomorska Akademia Pedagogiczna, Instytut Matematyki, Poland
<i>Hossam A. H. Fahmy</i> , Electronics and Communications Department, Cairo University, Egypt	<i>Grażyna Jackowska</i> , PSOOU, Poland
<i>Deimantas Galčius</i> , VTeX Ltd., Lithuania	<i>Bogusław Jackowski</i> , BOP s.c., Poland
<i>Agnieszka Gdaniec</i> , Poland	<i>Paweł Jackowski</i> , GUST, Poland
<i>Johannes Große</i> , Institute of Physics, Jagiellonian University, Poland	<i>Arkadiusz Jasiński</i> , Uniwersytet Kazimierza Wielkiego, Poland
<i>Olga Große</i> , Poland	<i>Katarzyna Jaskólska</i> , Poland
<i>Atif Gulzar</i> , Center for Research in Urdu Language Processing, National University of Computer and Emerging Sciences, Pakistan	<i>Aleksandra Kaptur</i> , Uniwersytet Śląski, Instytut Matematyki, Poland
	<i>David Kastrup</i> , QuinScape GmbH, Germany
	<i>Jonathan Kew</i> , SIL International, England

Ewa Kmieciak, Zakład Hydrogeologii i Ochrony Wód
AGH, Poland
Jacek Kmieciak, Uczelniane Centrum Informatyki
AGH, Poland
Sabina Kmieciak, Poland
Mateusz Kmieciak, Poland
Ewa Koisar, Polskie Towarzystwo Mechaniki
Teoretycznej i Stosowanej, Redakcja JTAM, Poland
Adam Kolany, Uniwersytet Jagielloński, Wydział
Matematyki i Informatyki, Poland
Dorota Kolany, Poland
Michał Kolany, Poland
Harald König, DANTE e.V., Germany
Reinhard Kotucha, DANTE e.V., Germany
Siep Kroonenberg, Rijksuniversiteit Groningen,
Economische Wetenschappen, The Netherlands
Jano Kula, Czech Republic
Johannes Küster, Typoma GmbH, Germany
Ulrike Küster, Germany
Dag Langmyr, Department of Informatics, University
of Oslo, Norway
Jerzy Ludwichowski, Uniwersytet Mikołaja Kopernika,
Poland
Tomasz Luczak, TECHNODAT Sp. z o.o., Poland
Ewelina Luczak, Poland
Małgorzata Luczak, Poland
Wojciech Lukaszewicz, Global Village Sp. z o.o.,
Poland
Włodzimierz J. Martin, Poland
Mojca Miklavc, Slovenija
Ross Moore, D-MTH (SAM), Switzerland
Grzegorz Murzynowski, GUST, Poland
Janusz Nowacki, FOTO-ALFA, Fotokład
Komputerowy, Poland
Heiko Oberdiek, University Freiburg & DANTE e.V.,
Germany
Andrzej Odyniec, Macrologic S.A., Poland
Bram Otten, The Netherlands
Robyn Parkinson, Switzerland
Karel Píška, Institute of Physics, Academy of
Sciences, Czech Republic
Norbert Preining, Università di Siena, Italy
Arkadiusz Prokop, Polskie Towarzystwo
Informatyczne, Poland
Arthur Reutenauer, École Normale Supérieure, France
Jan Ryćko, V Liceum Ogólnokształcące, Gdańsk,
Poland
Joanna Ludmiła Ryćko, Humboldt Universität
zu Berlin, Germany
Marek Ryćko, Wydawnictwo Do, Poland
Volker RW Schaa, DANTE e.V., Germany
Martin Schröder, pdfT_EX team, Germany
Petr Sojka, Masaryk University, Faculty of
Informatics, Czech Republic
Vytas Statulevičius, VTE_X Ltd., Lithuania
Péter Szabó, Budapest University of Technology and
Economics, Department of Computer Science and
Information Technology, Hungary
Ewa Szelatyńska, Poland
Jolanta Szelatyńska, Uniwersytet Mikołaja Kopernika,
Uczelniane Centrum Informatyczne, Poland
Hàn Thế Thành, River Valley Technologies, Germany

Andrzej Tomaszewski, GUST, Poland
John Trapp, England
Radosław Tryc, RTC Radosław Tryc, Poland
Ulrik Vieth, DANTE e.V., Germany
Stanisław Walczak, GUST, Poland
Zofia Walczak, Uniwersytet Łódzki, Wydział
Matematyki, Poland
Stanisław Wawrykiewicz, Poland
Ferenc Wettl, BME Institute of Mathematics,
Hungary
Marek Wójtowicz, Uczelniane Centrum Informatyki
AGH, Poland
Agnieszka Wójtowicz, Poland
Marta Wolińska, Instytut Biocybernetyki i Inżynierii
Biomedycznej PAN, Poland
Marcin Woliński, Instytut Podstaw Informatyki PAN,
Poland
Agata Wylot, Redakcja IO PAN, Poland
Jakub Zdroik, Uniwersytet Gdański, Instytut
Oceanografii, Poland

Copyright © 2007 T_EX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the T_EX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

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

Biuletyn Polskiej Grupy Użytkowników Systemu T_EX (ISSN 1230-5630) is published by GUST.

Die T_EXnische Komödie (ISSN 1434-5897) is published by DANTE e.V.

This publication was distributed by the Czechoslovak T_EX Users Group to its members.

For web sites and information on all the T_EX user groups, see <http://tug.org/usergroups.html>. Please join the user group best for you!

The script typeface used on the covers and title page of this issue is Vassallo 1.0, created by Jefferson J. Vorzimmer, «bang the drum» graphics, with modifications by Bogusław Jackowski.

T_EX is a trademark of the American Mathematical Society.

Printed in Germany, December 2007.

EuroT_EX 2007 — Regular sessions

Saturday, April 28

- 10:00 *coffee*
chair: Volker RW Schaa
- 10:30 Jerzy Ludwiczowski *Conference Opening*
11:00 Jonathan Kew *X_YT_EX in T_EX Live and beyond*
11:30 Taco Hoekwater *Have no fear, MEGAPOST is here!*
12:00 Hans Hagen *LuaT_EX: messing around with tokens*
12:45 Joanna Ludmiła Ryćko *The T_EX Clinic*
- 13:00 *lunch*
chair: Jonathan Kew
- 15:00 Johannes Große *'MathPSfrag': creating L^AT_EX labels in Mathematica plots*
15:30 Siep Kroonenberg *'epspdf': easy conversion between PostScript and pdf*
16:00 Zofia Walczak *Graphics in L^AT_EX using TikZ*
- 16:30 *coffee*
chair: Petr Sojka
- 17:00 Norbert Preining *An experimental CTAN upload process*
for Jim Hefferon
- 17:30 Jean-Michel Hufflen *Introducing L^AT_EX users to XSL-FO*
18:00 Grzegorz Murzynowski *The 'gmdoc' bundle: a new tool for documenting (L^A)T_EX sources*
18:20 Grzegorz Murzynowski *The 'gmverse' package*
18:40 Marek Ryćko *Polishing typesetting blocks*
- 19:10 *dinner and bonfire*

Sunday, April 29

- chair*: Karel Horak
- 09:00 Andrzej Tomaszewski *Designing a book: with both pleasure and . . . fear*
10:00 Dorota Cendrowska *Enumerations as an interesting form of text appearance*
10:15 Karl Berry, Jerzy Ludwiczowski *GUST Font License: an application of the L^AT_EX Project Public License*
10:30 Jean-Michel Hufflen *MIBIB_TE_X: reporting the experience*
- 11:00 *coffee*
chair: Ross Moore
- 11:30 Jean-Michel Hufflen *Managing order relations in MIBIB_TE_X*
12:00 David Kastrup *Writing L^AT_EX documents with AUCT_EX in Emacs*
12:45 Péter Szabó *Automated DVD menu authoring with pdfL^AT_EX*
13:05 Norbert Preining *T_EX (Live) on Debian*
- 13:35 *lunch*
chair: Klaus H^oppner
- 15:00 Atif Gulzar, Shafiq-ur Rahman *Nastaleeq: a challenge accepted by Omega*
15:30 Amir M. S. Hamdy, *Parameterized Arabic font development for AlQalam*
Hossam A. H. Fahmy
- 16:00 Martin Schröder *pdfT_EX 1.40: What's new*
- 16:30 *coffee*
chair: Hossam A. H. Fahmy
- 17:15 Karel Horák *Those obscure accents . . .*
17:45 Hàn Thế Thành *Typesetting Vietnamese with VnT_EX (and with the T_EX Gyre fonts too!)*
18:15 Tomasz Luczak *L_YX: an editor not only for secretaries*
18:45 Tomasz Luczak *From wiki to T_EX*
- 19:05 *dinner*

Tuesday, May 1

- chair*: Alexander Berdnikov
- 09:00 Hans Hagen *ConT_EXt and OpenType: what kind of font system do we need*
09:45 Taco Hoekwater *Open Type fonts in LuaT_EX*
10:15 Hàn Thế Thành *Font-specific issues in pdfT_EX*
10:45 Grzegorz Murzynowski *L^AT_EX: a modification of the logo*

11:00	<i>coffee</i> <i>chair:</i> Hans Hagen	
11:30	Sam Guravage	<i>Confessions of a teenage T_EX user</i>
11:45	Jerzy Ludwichowski for Jonathan Fine	<i>MathTran — T_EX as a Web service</i>
12:15	David Kastrup	<i>‘qstest’, a L^AT_EX package for unit tests</i>
12:35	David Kastrup	<i>‘makematch’, a L^AT_EX package for pattern matching with wildcards</i>
12:55	David Kastrup	<i>Benefits, care and feeding of the ‘bigfoot’ package</i>
13:30	<i>lunch</i> <i>chair:</i> Taco Hoekwater	
15:00	Klaus H ^o ppner	<i>Creation of a PostScript Type 1 logo font with MT1</i>
15:45	Petr Sojka, Michal R ^u žička	<i>Single-source publishing in multiple formats for different output devices</i>
16:15	Péter Szabó	<i>Practical journal and proceedings publication on paper and on the web</i>
16:45	David Kastrup	<i>DocScape Publisher: a large scale project based on T_EX</i>
17:15	<i>coffee</i> <i>chair:</i> Hàn Th ^e Thành	
17:45	Karel Píška	<i>Procedures for font comparison</i>
18:05	Karel Píška	<i>Comments and suggestions about the Latin Modern fonts</i>
18:45	Janusz M. Nowacki	<i>Cyklop: A new font family</i>
19:15	<i>dinner</i>	

Wednesday, May 2

	<i>chair:</i> Hans Hagen	
09:00	Paweł Jackowski	<i>T_EX: Beauties and oddities</i>
09:45	Ross Moore	<i>Advanced mathematics features, for PDF and the Web</i>
10:30	<i>coffee</i>	
11:00	Arthur Reutenauer	<i>A brief history of T_EX</i>
11:45	Ulrik Vieth	<i>Overview of the T_EX history archive</i>
12:15	Bogusław Jackowski, Jerzy B. Ludwichowski, Janusz M. Nowacki	<i>Recent advances in LUGs’ font projects</i>
13:00	Marek Ryćko	<i>Data structures in T_EX</i>
13:45	Jerzy Ludwichowski	<i>Conference Closing</i>
14:00	<i>lunch</i>	

EuroT_EX 2007 — Workshops

Saturday, April 28

12:00	Andrzej Tomaszewski	<i>Designing graphical signs and logotypes</i>
-------	---------------------	--

Sunday, April 29

9:00	Grażyna Jackowska	<i>Handmade paper: A mixture of handcraft, art and fun</i>
------	-------------------	--

Conference outing to Toruń and Chełmno

Monday, April 30

Highlights:

- visit to the Wojewódzka Biblioteka Publiczna–Książnica Kopernikańska (“District Public Library–Copernican Library”),
- a guided tour through the Toruń city center,
- a quick stroll through the city center of Chełmno,
- conference dinner and live performance of the “Schola Teatru Wiejskiego Węgajty” band, led by Kasia Jackowska.

EuroBachTeX 2007: Paths to the Future

The XVIIIth European TeX conference was a double double—joint with the XVth BachTeX, the yearly GUST (the Polish TeX users group) conference, and co-organized by CSTUG (the Czechoslovak TeX users group). It took place in Bachotek, the traditional place for GUST conferences, from April 28 until May 2, 2007, hence its other name: EuroBachTeX 2007.

By publishing these proceedings we follow the “new tradition” started in 2005 by DANTE and Gutenberg with the Pont-à-Mousson EuroTeX proceedings, of trying to reach with the material presented the widest possible audience. The “Paths to the future” theme of this conference was coined to account for the flurry of recent developments: XeTeX, LuaTeX, TeX Gyre and Latin Modern fonts, continued efforts by the $\epsilon\lambda$ TeX team, new versions of pdfTeX, METAPOST (the future MEGAPOST?), MiBibTeX (a prospective successor to BibTeX) as well as new efforts by the Arab-speaking TeXies to enhance their fonts and typesetting facilities. A special mention should go to the efforts of Jim Hefferon and the other members of the CTAN team who quietly and behind the scenes are working on enhancements to this TeX gold mine.

We think that these developments should become known all over the TeX world. Achieving this ambitious goal wouldn't be possible without extensive cooperation. First let us mention the authors—they replied favorably to the Calls for Papers so we had all the above-mentioned efforts presented at the conference. When we announced the intention to publish this volume, the authors further took the opportunity to revise their papers, and updated them with the most recent information. Then there was the effort of the marvelous *TUGboat* editing team, Barbara Beeton and Karl Berry, who took care of the proofreading and other editorial and typesetting chores. Bogusław Jackowski, supported by Karl, has done a splendid job of designing the cover and the photo pages (more on that below). Had not Karl Berry pushed the remaining members of the Editorial Board, we would not have been able to meet the December deadline. The printing itself and the complicated shipping logistics were arranged by Volker RW Schaa and Klaus Höppner from DANTE. Perhaps the most important factor in this undertaking is that so many user groups (DANTE, TUG, UK-TUG, GuIT, CSTUG, MaTeX and GUST) expressed their interest and ordered copies of this volume for their

members—the print run is in excess of 5000.

To stress that this volume originates in Europe, and to follow the “new tradition”, the cover colors are the same as the EuroTeX 2005 volume—(eu)blue and (eu)yellow from the Europe flag.

Now a little about the volume content and what was happening during the conference. We hope that thanks to the efforts of the authors (about 50 papers) the goal of giving an overview of what is happening in the TeX universe and what lurks in the future was achieved. We also wanted to give those who were not present at the venue a glimpse of the atmosphere, hence the color photo section. It was composed by Bogusław Jackowski, who chose a selection of photographs from shots taken by Jacek Kmieciak, Andrzej Odyńiec, Volker RW Schaa, Ulrik Vieth and Jakub Zdroik. A far larger collection of photos is available at the GUST conference site, <http://www.gust.org.pl/BachTeX/EuroBachTeX2007>, in the *gallery* section. The *movies* section hosts movies taken by Jan Ryćko, and in the *presentations* section slides for some of the sessions are available.

Two workshops were held. A great success was the “Handmade paper: A mixture of handcraft, art and fun” workshop run by Grażyna Jackowska—a glimpse of it can be had by reading the color-illustrated report following. Andrzej Tomaszewski also gave an excellent workshop on designing graphical signs and logotypes—a smaller, self-selected group of interested participants picked Andrzej's brain to their benefit.

There were two surprise events. The presentation by Sam Guravage, the youngest ever presenter at a EuroTeX conference was distinguished as the best presentation of the conference and he received a special diploma—please read the report in this volume. A surprise even to the organizers was Janusz M. Nowacki's exhibition of his early photographs—a mirror of his artistic soul, so well expressed in his font achievements.

The hard conference work was interrupted by a day's outing to the not-so-far-away cities of Toruń and Chełmno. In Toruń the participants visited the District Public Library—Copernican Library with a special presentation of their incunabula collection (see the *gallery* section of the conference site), followed by a guided tour through the medieval city center, a monument listed by the UNESCO World Heritage. On to Chełmno, a nice, small city, spread out on seven hills—a quick stroll through the city

center with many medieval traces followed by the conference dinner at the local Karczma Chelmińska (Chelmno Hut). The highlights (see *gallery*) were good food, the conference cake and dancing to the performance of the “Schola Theatrum Węgajty” band led by Kasia Jackowska, Jacko’s daughter. This tiring day of rest ended in Bachotek at about 3 am.

A user’s report on the conference, written by Michael Guravage, was published in the recent issues of NTG’s MAPS, DANTE’s Technische Komödie, and TUG’s TUGboat, therefore it is not reprinted here.

A special mention should go to Kasia Jackowska for designing the conference logo (cf. the cover and the conference site for the color versions) and Andrzej Tomaszewski for the conference poster, which was prepared in the three conference languages: English, Czech and, of course, Polish.

Special thanks go to all the members of both the Program and Organizing Committees (the names are listed elsewhere in this volume), the IT team (Marek Czubenko and Janusz Gumkowski, who enjoyed the support by Harald Köenig), the session

chairs (their names are also listed elsewhere in this volume) for keeping us on track with a very tight schedule, the official Conference Still Photographers (Andrzej Odyniec, Jacek Kmiecik), the Conference Chief Cinematographer (Jan Ryćko) and even more special thanks to Jolanta Szelatynska, the one-woman institution of GUST conferences, and all her helpers.

The conference would not be possible without the aid of the sponsors, as they made it financially viable. The list of sponsors is given elsewhere in the volume. Very many thanks!

Last but not least: the participants. They came in great number, brought their spouses and children and seemed to enjoy the conference despite some organizational slips and the not-so-good-as-usual weather.

Come back to future TeX conferences, with BachTeX 2008 being no exception. In the meantime: enjoy the volume and ... Happy TeXing!

*Jerzy Ludwichowski and Petr Sojka
December 2007*

TeX contra TeX—Is the future now?

➤ Invitation to BachTeX 2008 ◀

GUST—the Polish TeX users group
invites you to the jubilee XVIth BachTeX conference
to be held at the usual location in Bachotek
near Brodnica, Poland, from April 30 to May 4, 2008.

<http://www.gust.org.pl/BachTeX/2008>

And an invitation to TUG 2008

TUG invites you to its 2008 annual conference as well,
celebrating TeX’s 30th birthday!

University College Cork • Cork, Ireland
21–24 July 2008 • <http://tug.org/tug2008>











Confessions of a teenage \TeX user

Sam Guravage

Literate Solutions, Loenen

Holland

guravage (at) literatesolutions (dot) com

Abstract

In this presentation I will tell the audience a bit about how I use \TeX for my everyday school work. I will explain what tools I use, how my father convinced me to try it, and finally if I like it or not. Of course there will be plenty of time for questions of how it came thus far.

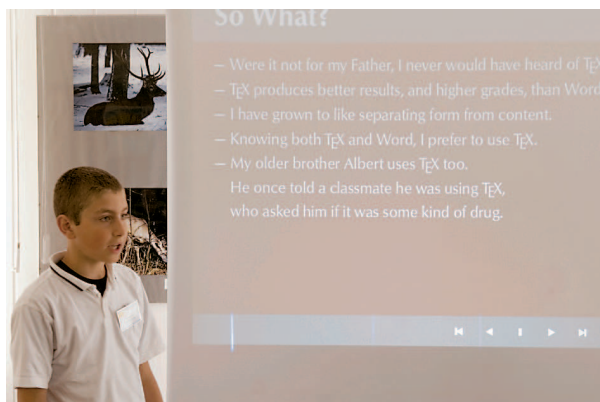


Figure 1: Sam Guravage—the youngest ever to present at Euro \TeX

Unlike the other articles in these proceedings, this note (except for the abstract) was written not by the author but by one of the members of the audience, who had the happy opportunity to be there to see Sam's presentation (fig. 1),¹ the youngest ever to give a talk at a Euro \TeX conference, a representative of those who are just starting on the "Paths to the Future".

In a very matter-of-fact, straightforward and confident (but not overly so) manner (fig. 3, right) he told us why and for what he uses \TeX , what he finds easy and what hard to do with \TeX and how he benefits from using \TeX .

At the end of the presentation we received an insight into the future of the young generation of the Guravage family: they have the potential to become \TeX dealers. Those who are puzzled are encouraged to look at Sam's slides, available from the "presentations" folder at <http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007>. What I especially

¹ Photographs: 1, 2, and the right part of 3 are by Andrzej Odyniec; the left part of 3 is by Volker RW Schaa.

like about them is that they were completely—form and content—done by Sam.

"Next slide, please": Sam was watched by his, perhaps a bit skeptical, but surely very proud father, who also gave him a hand (fig. 2).

During conference closing, Sam was awarded a diploma for the best presentation (fig. 3, left).

We surely hope that he will be returning and others will follow.



Figure 2: Under parental supervision.



Figure 3: Left: with the diploma; right: self-confident

Handmade paper: A mixture of handcraft, art and fun

Grażyna Jackowska

PSOUU, Poland

grazynajackowska (at) wp dot pl

Abstract

The idea of a paper making workshop was devised to counterbalance the serious technical discussions. It turned out that many were interested: both adults and “conference children” wanted to participate in this adventure (fig. 1).



Figure 1: Participants waited patiently in a queue; children, youngsters and grown-ups were interested in touching the *in statu nascendi* paper.

First: thanks to Jacek Kmiecik for the photographs!

1 How we did it

The aim was to walk through almost all stages of the technological process — from making the pulp,



Figure 2: Helping hands: the pulp was scooped with a wire screen or sieve.

scooping it with with a sieve, removing excess water with a press (figs. 2 and 3), drying the paper sheets, up to putting the product to use.

The work was accompanied by stories about the history of paper — its way from China through the Arabic empire to Europe and about the improvements in fabrication technology.

2 How we used it

One person wrote a letter (fig. 4), another a solemn memorial — a sheet of such paper was used to prepare a diploma for Sam Guravage as an award for the best presentation (fig. 5).

3 Why we did it

Also important was coming to grips with the creative process, i.e., composing a decorated paper sheet with such additions as dyes, grass leaves, flowers and even ... banknotes (fig. 6). On one of the sheets a four-leaf clover grows — will it bring luck to somebody?

The unaided composing of the paper-picture seems to have been the greatest attraction of the workshop — all the time new students were turning up — “I will yet do this so... or perhaps so...”



Figure 3: Strong hands: excess water was removed using a hand press.



Figure 4: Nice paper needs thorough concentration to write a nice letter...



Figure 5: Sam Guravage receives his handmade diploma from Jerzy Ludwichowski.

The results were presented at a post-workshop exhibition (fig. 7)—it can be seen that amazingly varied ideas emerged despite the simple means, and the number of works created in such a short time bespeaks that we are much in need of such “amusements”—of such creativity written even with the



Figure 6: “Here are the colors of nature so that you see what I see” reads the text in the Arabic script by Hossam A.H. Fahmy. Hossam explained further that: “The blue/green background represents the lake and the trees. The flowers and the leaves are for the blooming spring around us and for a hope for a fruitful T_EX/METAFONT future”—perfect touch!



Figure 7: The variety of the ideas was impressive, indeed...

smallest “c”, even if our everyday occupation is a so-called serious one. Or, perhaps, besides for a playful moment, something will remain in one of the type-setting souls? A broader, different view of the mysterious charm of paper as a medium not only for history, tradition, information, but also beauty?

Designing a special book: With both pleasure and . . . fear

Andrzej Tomaszewski
Warszawa, Poland
andrzej (at) stegny dot 2a dot pl

Abstract

I will present the various conditions and limitations which should be taken into account when designing and producing an album-like book: contents, typography, materials. The work was for a company which never before had anything to do with publishing: the Municipal Water Supply and Wastewater Treatment Company — master of the life arteries of greater Warsaw. The publication is a jubilee edition, not for sale, although it may later appear in second-hand bookshops.

Warsaw's *Filtry*, that is, “Miejskie Przedsiębiorstwo Wodociągów i Kanalizacji” (Municipal Water Supply and Wastewater Treatment Company) is celebrating its 120th anniversary. The company's history started during the presidency of the highly esteemed Sokrat Starynkiewicz, an artillery general of the Russian tsar, with a time of activity of an excellent team of designers and builders which was assembled and directed by William Lindley, an English engineer.

A paradox of history! We are now looking from a different perspective at the occupying forces which built in Warsaw a powerful fortress for the Russian garrison, in whose dungeons Polish patriots suffered and whose walls were covered by gallows. The tsar's commissioner, Starynkiewicz, simply fell in love with

the occupied city! Even today he is regarded as its best president, a president who had a vision of a modern city agglomeration and implemented this vision unceasingly. The jubilee was an occasion to produce a publication of over 200 pages: “Dla dobra publicznego. 120 lat Wodociągów Warszawskich” (For the public welfare. 120 years of the Warsaw Waterworks), an album with a complicated structure, edited by Piotr Stankiewicz, and designed and typeset by myself. The work contains — except for text materials — mainly archival illustrations, often unique, coming from the company's collection, Warsaw museums and libraries, and in the contemporary era — mostly — photographs by Krzysztof Kobus. The publisher received high-quality Post-

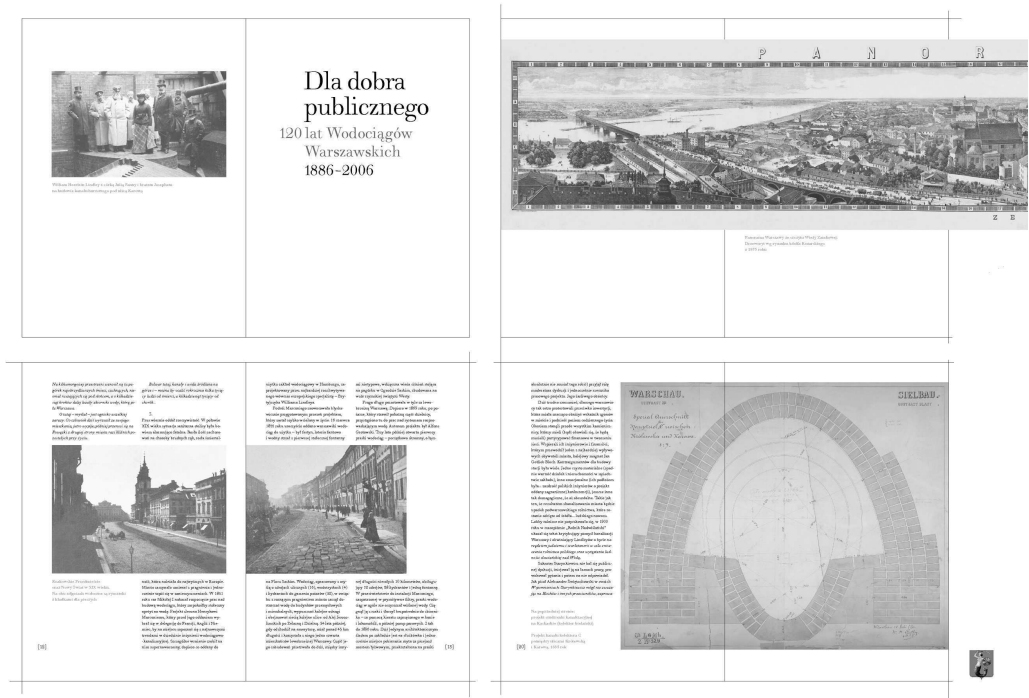


Figure 1: One of the book's design sketches showing the title page and the fly-leaf glued into the spine with a reproduction of a wood-engraved panorama of XIV century Warsaw. The trim size dimensions are 226 by 320 mm.

Script files for the printing process from Katarzyna Ciemny.

The typesetting was done in the Walbaum typeface (from the Linotype Library font collection). The font, popular in the XIX century, was designed by a German type founder from Weimar. It perfectly corresponds to the spirit of the epoch during which the Warsaw waterworks and sewage system were built. The breaking-up of pages and illustrations follows a quite simple modular setup which was not followed very strictly — in a few places aesthetic reasons called for modifications.

The archival illustration materials caused considerable difficulties. I had to reproduce lithographs, chromolithographs, guaches, wood engravings, original and printed photographs (all of different resolutions). The easiest were — of course — the contemporary digital photographs made by a professional photographer. Each of these types of images required a different scanning procedure and adherence to a different color regime. By the way, every printer knows that the qualities and “warmth” of traditional graphic techniques cannot be mimicked absolutely by a CMYK reproduction.

This impressive, and very difficult to produce, publication was printed and bound by the Olsztyńskie Zakłady Graficzne — in a very short time and while retaining the highest quality. It was made possible only by an exceptional devotion of the printers from Olsztyn.

Several kinds of paper were used for the book. For the historical section, where the resolution was 175 lpi, Lessebo Design Ivory 120 g was applied. To accommodate a set of archival photographic materials from the Warsaw Uprising, a sheet of composition enclosed in a cover of Owl Grey (Card) from the Kaskad catalog was inserted between the historical and contemporary parts. The chapter on the sewers’ role in the Warsaw Uprising was printed in duotone with Pantone Black 2C and 409C Gray. The contemporary part was printed — also with 175 lpi resolution — on chloride free Media Print Matt 150 g paper.

The historical part opens with a fly-leaf of the Warsaw panorama printed on Century Free Life Velum White 100 g. In two places — between printing sheets — glued inserts on quasi-parchment Golden Extra White 110 g tracing paper with XIX century design drawings were placed.

The album, printed with a Speedmaster CD102 printing machine, is bound in a natural brick-red cloth, Brillant, made by Van Heek Schoco. Endpapers were printed on Fabrizia Brizatto 120 g from Cartiere Fabriano. The front cover has an insert glued precisely into an inset rectangle. The title on the spine is stamped with a golden foil.

One arrives at the full picture of the difficulty of the binding task if one realizes that such an already-rarely encountered binding was additionally complicated by two parchment paper glued inserts, two

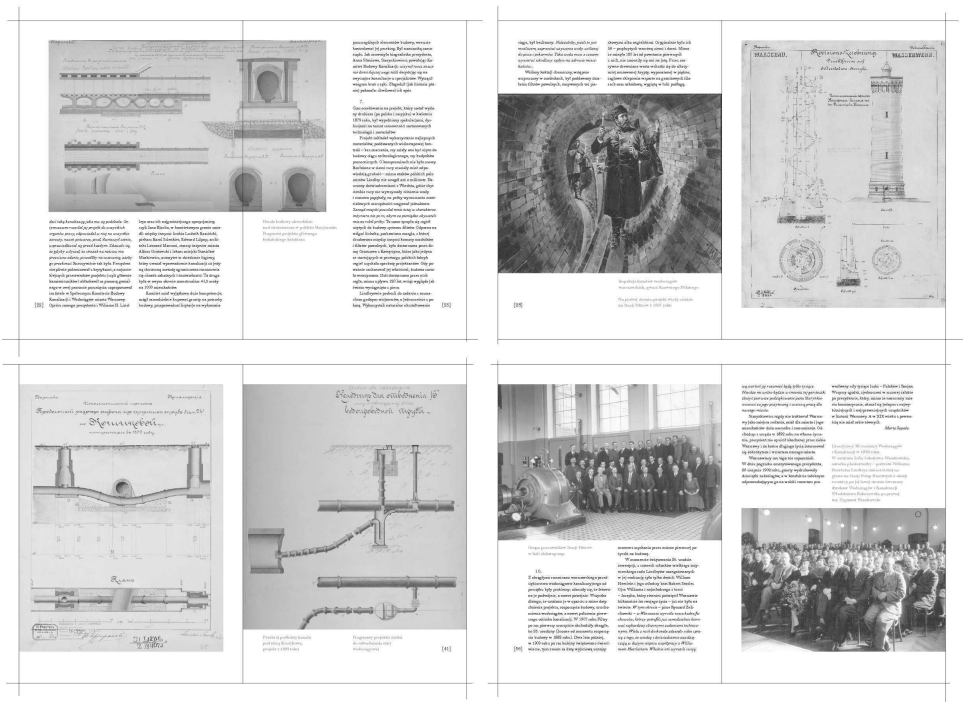


Figure 2: Design sketch showing how illustrations are to be placed in the historical part of the book.



Figure 3: The design of the part devoted to the role of the sewers in the Warsaw Uprising. The insurgents and civilians used them as emergency routes between streets and even between different quarters of the city.

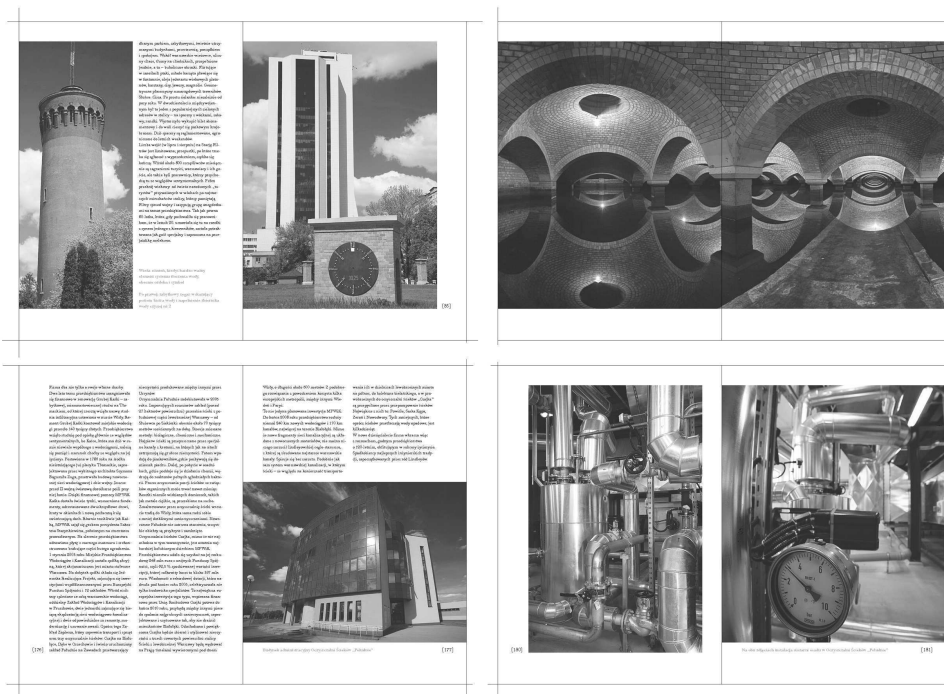


Figure 4: A fragment of the design of the contemporary part. The inner city of Warsaw is the only place in Europe where modern water and sewage installations work next to XIX century engineers' installations deployed full-scale.

fly-leaves (one of which is in an atypical format) as well as an internal cover of one of the chapters.

And the additional complication was . . . time— from the technological point of view— not enough of it. The jubilee celebrations could not be delayed. Descendants of William Lindley came from all over the world. The floors of the Royal Castle in Warsaw had been polished, Kayah¹ was all geared up for a concert at the Wisła bank. And what . . . no album? Happily, the book binding team valiantly managed to complete this complicated operation in time.

To those present at Bachotek, I showed two

copies of this bibliophile piece. Not many people are going to have a chance to get to know the precision accomplished by the editors, graphic artists, photographer and typesetters from Olsztyn— the print run was a little over one thousand copies and the book was not for sale. Perhaps in the future it might show up in the secondary circulation but no doubt already as a rare and valuable item.

(During the presentation, PDF versions of the objects comprising the book were also shown and technical aspects of the realization were discussed. The slides are available from the “presentations” folder at <http://www.gust.org.pl/BachTeX/EuroBachTeX2007>.)

¹ Kayah (Katarzyna Rooijens)— a Polish pop singer.

Enumerations as an interesting form of text appearance

Dorota Cendrowska

Polsko-Japońska Wyższa Szkoła Technik Komputerowych, Warszawa
dorota dot cendrowska (at) pjwstk dot edu dot pl

Abstract

The paper presents classical types of enumerations and the ways to create them. The rules for single- and multi-sentence as well as nested enumerations are discussed. Even if (L^A)T_EX is treated as a king in the science and technical departments of higher education institutions, we will show that sometimes the king's gown might be in need of mending. We will discuss methods that are available for changing the appearance of enumerations in their wide and varied usage (printed text, multimedia presentations).

Introduction

One can successfully defend the proposition that one needs to know quite a lot to typeset a good-looking text using an ordinary word processor, and also that one has to learn quite a lot to do things with L^AT_EX in one's own way. As a consequence, documents typeset with L^AT_EX usually have a friendly, familiar, look. This feature and the fact that no other tool exists which typesets mathematics better than the T_EX family makes (L^A)T_EX an unquestioned king of technical departments at universities. As is well known, one doesn't question the king. If something was typeset with L^AT_EX then, by default, it is typeset well.

Although trying to improve something good might not turn out to be the best idea, in the rest of this article we will try to find "something better" using the example of enumerations. Nowadays it is almost impossible not to stumble upon enumerations at almost every "corner". We will describe "classical" rules of typesetting enumerations and then discuss their relationship to the enumerations readily available with L^AT_EX.¹

1 The classics of enumerations

Several centuries of typesetting experience brought with it a typesetting canon. Typography, the "art of printing", has quite a lot to offer in a seemingly narrow area as enumerations. Enumerations (or lists) may be categorized depending on the nature of the "constituting points".

If consecutive items constitute a single sentence (Example 1) then we have a single-sentence enumeration. In such a case the enumeration can be typeset properly in several ways: treating it as a single text

paragraph (Example 1) or putting each item into a new line (Examples 2–4).

According to the rules of annotating consecutive items, lower case letters or Arabic numerals are used, after which the closing bracket follows (Examples 1–3). Periods should not be used as, except for abbreviations, they denote the end of a sentence and we have here a single-sentence enumeration. When each of the items is placed on a separate line, an em-dash might be used (Example 4).

(...) you always have at least four choices. . . a) the two opposites and then b) the middle ground and c) "taken under further contemplation."²

Example 1: Single-sentence enumeration

(...) you always have at least four choices. . .
a) the two opposites and then
b) the middle ground and
c) "taken under further contemplation."²

Example 2: Single-sentence enumeration

(...) you always have at least four choices. . .
1) the two opposites and then
2) the middle ground and
3) "taken under further contemplation."²

Example 3: Single-sentence enumeration

¹ Only Polish typography rules will be discussed, but they might be of general interest.

² Clarissa Pinkola Estés, *Women Who Run With the Wolves: Myths and Stories of the Wild Woman Archetype*, Ballantine Books, 1997.

(...) you always have at least four choices. . .
 — the two opposites and then
 — the middle ground and
 — “taken under further contemplation.”²

Example 4: Single-sentence enumeration

It is enough to concentrate a little to see that the seemingly simplest things and events from everyday experience can awaken a feeling of an impenetrable mystery:

- 1) time, freedom, existence, space;
 - 2) cause, awareness, matter;
 - 3) number, love, “I”, death.³
-

Example 5: Commas and single-sentence enumerations

In single sentence enumerations the period is used only once, at the end of the last item of the enumeration. The preceding items are terminated with commas or — if the items contain many commas already — with semicolons. This rule is illustrated by Example 5.

If the contents of an enumeration cannot be expressed in the form of one sentence, then we have to deal with a multi-sentence enumeration. Each item takes the form of an arbitrary number of full, individual, sentences. To annotate the consecutive items of such an enumeration, numbers or uppercase letters followed by a period should be used. The period informs that “what will immediately follow is the beginning of a sentence.” Important to note is that for both single- and multi-sentence enumerations the sentences (not their annotations) begin at the same distance from the page edge. A multi-sentence enumeration, typeset in the classical way, is given in Example 6.

Consider this:

A. Rafer Johnson, the decathlon champion, was born with a club foot.

B. Winston Churchill was unable to gain attendance to the prestigious Oxford or Cambridge universities because he “was weak in the classics.”

C. In 1905, the University of Bern turned down a doctoral dissertation as being irrelevant and fanciful. The young physics student who wrote the dissertation was Albert Einstein, who was disappointed but not defeated.⁴

Example 6: Multi-sentence enumeration

³ Leszek Kołakowski, *Mini wykłady o maxi sprawach*, seria druga, Znak, Kraków 1999.

⁴ Jack Canfield, Mark Victor Hansen, *A 3rd Serving of Chicken Soup for the Soul*, Health Communications, 1996.

Enumerations might be *nested*, which means that an enumeration becomes an item of a different, higher ordered item. One can thus talk about a hierarchy of enumerations which should be clearly distinguished in the typeset text so that no ambiguity arises as to which “matrioshka” is put into which. The following rule applies when annotating items of a multilevel enumeration: When moving from the main (outermost) enumeration to the most nested enumeration, i.e., from the biggest to the smallest “matrioshka”, the consecutive items are labeled with

- uppercase roman numerals,
- uppercase letters,
- Arabic numerals,
- lower case letters.

Some of the labeling types might be omitted but their order should be kept. The period or closing bracket is used depending on whether the enumeration is single- or multi-sentence.

L^AT_EX allows up to four levels of enumerations of the type *itemize* or *enumerate*. The typesetting rules say that — if possible — nested enumerations should be limited to two levels. Example 7 illustrates multilevel enumerations.

- A. Outermost enumeration — 1st item:
 - 1) first item of a subordinate single-sentence enumeration,
 - 2) second item,
 - 3) third item,
 - B. Outermost enumeration — 2nd item:
 1. An item of a nested enumeration, which is a multi-sentence enumeration and may consist of an arbitrary number of sentences:
 - a) first element,
 - b) second element,
 - c) third element.
 2. Second item.
 3. Third item.
 - C. Outermost enumeration — 3rd item.
 - D. Outermost enumeration — 4th item.
-

Example 7: Multi-level enumerations

2 Tradition and L^AT_EX

In the previous section we sketched the traditional Polish conventions for typesetting enumerations. Perhaps the Anglo-Saxon tradition is completely different (what are the rules?), hence the default numbering and bulleting rules in the L^AT_EX environments

-
- Level 1, item 1
 - Level 2, item 1
 - * Level 3, item 1
 - Level 4, item 1
-

Example 8: An enumeration based on the \LaTeX *itemize* environment.

1. Level 1, item 1
 - (a) Level 2, item 1
 - i. Level 3, item 1
 - A. Level 4, item 1
-

Example 9: An enumeration based on the \LaTeX *enumerate* environment.

- item 1,
 - item 2,
 - item 3.
-

```

\renewcommand{\labelitemi}{---}
\begin{itemize}
  \item item 1,
  \item item 2,
  \item item 3.
\end{itemize}
```

Example 10: “The classics” in the \LaTeX environment *itemize*

enumerate and *itemize* — Examples 8 and 9. Anyway, if we treat the rules described above as a classic ensemble, then in principle we will feel properly dressed wearing it for any occasion. It suffices to redefine some internal \LaTeX commands to get the desired look of the labels (Examples 10 and 11).

3 Classics, \LaTeX and tiny details

A paragraph is a self-contained unit of text constituting a logical entity, “indivisible” from the point of view of the information conveyed; hence it is obvious that the following information should make exactly one paragraph:

Remember that whichever day of the week it is you always have the following choice for supper: a) to eat or b) not to eat. For breakfast you have no choice: you must eat it!

The same text formatted as a single-sentence enumeration (Example 12) confuses the reader by making the impression that the consecutive lines do not have much in common. This is caused by too much space between the consecutive items. Moreover, because the vertical space between the preceding paragraph and the enumeration is exactly the same as

-
- I. Level 1, item 1
 - A. Level 2, item 1
 1. Level 3, item 1
 - a. Level 4, item 1
-

```

\renewcommand{\labelenumi}{\Roman{enumi}.}
\renewcommand{\labelenumii}{\Alph{enumii}.}
\renewcommand{\labelenumiii}{%
  {\arabic{enumiii}.}
}
\renewcommand{\labelenumiv}{\alph{enumi}.}
\begin{enumerate}
  \item Level 1, item 1
  \begin{enumerate}
    \item Level 2, item 1
    \begin{enumerate}
      \item Level 3, item 1
      \begin{enumerate}
        \item Level 4, item 1
      \end{enumerate}
    \end{enumerate}
  \end{enumerate}
\end{enumerate}
```

Example 11: “The classics” in the \LaTeX environment *enumerate*

Remember that whichever day of the week it is you always have the following choice for supper:

- to eat or
- not to eat.

For breakfast you have no choice: you must eat it!

Example 12: A (confusing) single-sentence enumeration presented with standard spacing

between the enumeration and the following paragraph, one may be under the impression that the items of the enumeration have no logical relation to any of these paragraphs — each item of the enumeration seems to live its own life — but this is not so. We could live undisturbed by such a tiny detail, but — quietly — would prefer the result to be presented as in Example 13. (The exact spacing adjustment will vary with the class used.)

The *enumerate* and *itemize* environments have no means to take into account if the typeset enumeration is single- or multi-sentence because they simply “don’t know” anything of this nature. Therefore, one should take care to produce a less confusing spacing of the items. The scheme presented in Example 15 concerns itself with lengths defined in \LaTeX , at least some of which we can modify in the *enumerate* and *itemize* environments, just before the first item.

Remember that whichever day of the week it is you always have the following choice for supper:

- to eat or
- not to eat.

For breakfast you have no choice: you must eat it!

Example 13: A single-sentence enumeration as an entity

```
Remember that whichever day of the week
it is you always have the following
choice for supper:\vspace{-.6ex}
\renewcommand{\labelitemi}{---}
\begin{itemize}
\setlength{\parskip}{0ex}
\setlength{\itemsep}{0ex}
\item to eat or
\item not to eat.
\end{itemize}\vspace{-.6ex}
For breakfast you have ...
```

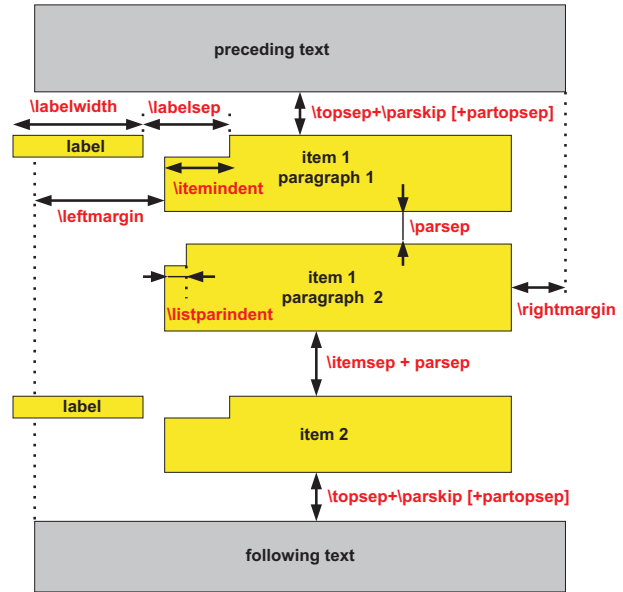
Example 14: How to typeset Example 13

An analysis of the code presented in Example 14 and a comparison with the output in Example 15 leads to the conclusion that the scheme of the meaning of the various lengths within the enumeration environment, although copied by many sources (also by [1]), is not very well in sync with the current practice. This one tiny detail should be remembered.

4 “Modern” enumerations

The classical methods of typesetting enumerations work well for paper publications. They are completely unsuitable for multimedia presentations, flyers or technical documentation. Such creations are typeset so as to facilitate quick access to various portions of text which, similar to dictionary entries, are self-contained entities.

Accordingly, the nature of enumerations is different. Spacing is more important and the same is true of the shape and position (with respect to the contents of the enumeration) of the symbols used to label the consecutive items—these are usually just entries. The nature of the symbols has also changed—they play a different role than in the classical enumerations. They are now rather “road signs” saying: “here is the beginning of the information which might be important for you.” Therefore they should be readable, of a proper size and, if possible, in a color contrasting with the color of the item’s



Example 15: Enumerations and L^AT_EX’s lengths

text (especially so in presentations).

It seems that the creator of L^AT_EX had a pre-cognition of the direction the world would roll. To create a “modern” enumeration, not yet covered by any norm, it suffices to change the labeling of the items. And so, Example 6, typeset to the classical rules, might be modified by changing the spacing and the way items are aligned (standard L^AT_EX enumeration)—Example 16—or by also modifying the item label—Example 17. Example 18 is given to show that we do not need to throw away all of the features of the classical enumerations, especially in brochures or flyers.

Summary

The world moves on, we are changing, enumerations change as well. It remains to wait for the day when an announcement found on our doormat will make us happy instead of dandering up—to our joy, the leaflet will be typeset with L^AT_EX ...

References

- [1] Kopka H., Daly P. W.: *A Guide to L^AT_EX*, 4th Edition, Addison-Wesley, 2003
- [2] Chwałowski R.: *Typografia typowej książki*, Helion, Gliwice 2002
- [3] Polish norms: PN-78/N-01222/03, BN-76/7440-03

Consider this:

- A. Rafer Johnson, the decathlon champion, was born with a club foot.
- B. Winston Churchill was unable to gain attendance to the prestigious Oxford or Cambridge universities because he “was weak in the classics.”⁴

Example 16: Multi-sentence enumeration differently

Consider this:

- ▶ Rafer Johnson, the decathlon champion, was born with a club foot.
- ▶ Winston Churchill was unable to gain attendance to the prestigious Oxford or Cambridge universities because he “was weak in the classics.”⁴

Example 17: Multi-sentence enumeration differently

■ *First year:* Sugar, I’m worried about my little baby girl. You’ve got a bad sniffle. I want to put you in the hospital for a complete checkup. I know the food is lousy, but I’ve arranged for your meals to be sent up from Rossini’s.

■ *Second year:* Listen, honey, I don’t like the sound of that cough. I’ve called Dr. Miller and he’s going to rush right over. Now will you go to bed like a good girl just for me, please?

■ *Third year:* Maybe you’d better lie down, honey. Nothing like a little rest if you’re feeling bad. I’ll bring you something to eat. Have we got any soup in the house?

■ *Fourth year:* Look, dear. Be sensible. After you’ve fed the kids and washed the dishes, you’d better hit the sack.

■ *Fifth year:* Why don’t you take a couple of aspirin?

■ *Sixth year:* If you’d just gargle or something instead of sitting around barking like a seal.

■ *Seventh year:* For heaven’s sake, stop sneezing. What are you trying to do, give me pneumonia?⁵

Example 18: Enumeration—modern and “about life”

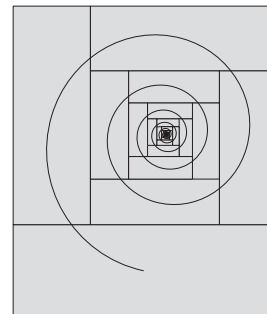
⁵ A. L. McGinnis, *The Romance Factor*, Harper & Row, 1990.

Five years after: Report on international T_EX font projects

Jerzy B. Ludwichowski
GUST, Poland
jerzy.ludwichowski@uni.torun.pl

Bogusław Jackowski
GUST, Poland
b_jackowski@gust.org.pl

Janusz M. Nowacki
Foto-Alfa, Grudziądz, Poland
J.Nowacki@gust.org.pl



Abstract

Two large font projects are being currently developed by the T_EX user groups: Latin Modern (begun in 2002) and T_EX Gyre. Both aim at greatly increasing the number of diacritical characters in the freely available collections of fonts.

1 Introduction

Since nearly the very beginning of their existence, Adobe's products have been accompanied with a basic collection of fonts, first containing only 13 components, then extended to 35: *ITC Avant Garde Gothic* (4 fonts), *ITC Bookman* (4 fonts), *Courier* (4 fonts), *Helvetica* (8 fonts), *New Century Schoolbook* (4 fonts), *Palatino* (4 fonts), *Times* (4 fonts), *ITC Zapf Chancery Medium* (1 font, italic), *Symbol* (1 font), and *ITC Zapf Dingbats* (1 font). Of course, they were only available under proprietary licenses.

The situation changed in 1996, when L. Peter Deutsch, then the developer and maintainer of the marvelous free PostScript interpreter Ghostscript, released in conjunction with URW++ a free collection of PostScript fonts, being a reasonable replacement for Adobe's 35 (see <http://tug.org/fonts/deutsch-urw.txt>).¹

Within a few years the collection was enhanced with Vietnamese and Cyrillic characters (by Hàn Thế Thành and Valek Filippov, respectively). Still, the repertoire of glyphs present in the fonts was insufficient to meet the needs of those who have to typeset in Latin-based languages.

2 The Latin Modern project

In Spring 2002 (hence the present title), several T_EX user groups launched the *Latin Modern Project*, with the aim of extending the Computer Modern fonts with a rich repertoire of Latin diacritical characters. This resulted in the Latin Modern collection of 92 fonts (72 text and 20 math) in the PostScript

¹ Actually, four URW++ free fonts appeared for the first time in Ghostscript 2.6.1 in 1993.

Type 1 and OpenType formats, (<http://www.gust.org.pl/projects/e-foundry/latin-modern>).

The warm reception of the project (despite various slips, though significantly reduced in number thanks to Karel Píška's continued efforts — see, e.g., his article in these proceedings) encouraged the user groups to make use of the Latin Modern experience to enhance the 35 basic PostScript fonts.

3 The T_EX Gyre project

Already in 1996, immediately after the release of the Ghostscript/URW++ fonts, the Polish T_EX users group, GUST, launched the *Qfonts* project, aimed at supplementing the relevant text fonts from the collection with a set of diacritical characters. The project was suspended after releasing a few fonts — other challenges became more important. However, with the success of the Latin Modern project, the groups supporting it decided to resume the *Qfonts* project, and indeed, greatly broaden its scope. The new incarnation became known as *T_EX Gyre*.

4 Licensing issues

Neither Adobe's nor URW++'s font names could be retained, due to legal issues. Therefore we have coined alternative names as follows:

Origin	PS name	TFM name (kernel)
ITC Avant Garde Gothic	TeXGyreAdventor	qag
ITC Bookman	TeXGyreBonum	qbk
Courier	TeXGyreCursor	qcr
Helvetica	TeXGyreHeros	qhv
Palatino	TeXGyrePagella	qpl
Times	TeXGyreTermes	qtm
New Century Schoolbook	TeXGyreSchola	qcs
ITC Zapf Chancery	TeXGyreChorus	qzc

(Note that the TFM names are, in a way, the legacy of the Qfonts project.)

Some licensing issues are unsolved. For example, it remains an open question whether the T_EX Gyre fonts can be released under the GUST Font License (GFL), a legal equivalent of the L^AT_EX Project Public License (LPPL) — see <http://www.gust.org.pl/projects/fonts/licenses/> and <http://www.latex-project.org/lppl/>.

The user groups are attempting to negotiate an agreement with the donor of the fonts, URW++.

5 The T_EX Gyre fonts

As of this writing, all 33 text fonts from the basic 35 fonts have been released (<http://www.gust.org.pl/projects/e-foundry/tex-gyre/>). Each contains nearly 1200 glyphs, including small caps, old style figures, Cyrillic and Greek. An exception is T_EX Gyre Chorus alias Zapf Chancery, which lacks Greek and small caps; incidentally, using capital forms of Chancery characters for typesetting whole words should be forbidden by law.

Note, however, that Cyrillic and Greek characters were included only provisionally (for the sake of uniformity and backward compatibility) and their quality must be improved as soon as possible. Only in T_EX Gyre Bonum (Bookman) is the Greek of better quality: with the kind permission of Apostolos Syropoulos and Antonis Tsolomitis, we imported the Greek glyphs from the Kerkis collection (<http://iris.math.aegean.gr/kerkis/>).

All fonts are available in OpenType and PostScript Type 1 formats. Of course, T_EX-oriented additions, that is, TFM files for various encodings plus the relevant MAP and ENC files, plus L^AT_EX support, are also provided. At present, the following encodings are available: CS (CSTUG), EC (Cork), L7X (Lithuanian), QX (GUST), RM (Regular Math or OT1), LY1 (Y&Y aka T_EX'n'ANSI), T5 (Vietnamese), TS1 (Text Companion for EC fonts) T2A, T2B, T2C (Cyrillic). The T_EX Gyre support for ConT_EXt is shipped separately with the ConT_EXt package.

Below are samples of the available fonts:

T_EX Gyre Adventor is available in Regular, SMALL CAPS, **Bold**, *Oblique*, and **Bold Oblique** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Bonum is available in Regular, SMALL CAPS, **Bold**, *Italic*, and **Bold Italic** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789**

23456789 0123456789 abcde ABCDE 0123456789 0123456789

T_EX Gyre Cursor is available in Regular, SMALL CAPS, **Bold**, *Oblique*, and **Bold Oblique** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Heros is available in Regular, SMALL CAPS, **Bold**, *Oblique*, and **Bold Oblique** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Heros Condensed is available in Regular, SMALL CAPS, **Bold**, *Oblique*, and **Bold Oblique** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Pagella is available in Regular, SMALL CAPS, **Bold**, *Italic*, and **Bold Italic** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Schola is available in Regular, SMALL CAPS, **Bold**, *Italic*, and **Bold Italic** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Termes is available in Regular, SMALL CAPS, **Bold**, *Italic*, and **Bold Italic** variants.

abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789 **abcde ABCDE 0123456789 0123456789 abcde ABCDE 0123456789 0123456789**

T_EX Gyre Chorus is available only in the Medium Italic variant:
abcde ABCDE 0123456789 0123456789.

6 Plans for the future

There is a lot to do: we plan to add the two remaining non-text fonts (for the sake of completeness), bugs reported so far have to be removed, Cyrillic and Greek are in need of improvement, kerning and hinting should be improved wherever needed, etc.

There are also much broader plans: to incorporate math into the T_EX Gyre collection — a truly serious task. It implies including math information in OpenType fonts and this requires research.

No fear that the T_EX Gyre spiral will ever reach its end. . .

Cyklop: A new font family

Janusz Marian Nowacki

ul. Śniadeckich 82 m. 46

86-300 Grudziądz

Poland

janusz (at) jmn dot pl; <http://www.jmn.pl>

Abstract

Cyklop, pl. Cyclops, (*gr. cyclos: round + ops: eye*) in Greek mythology, a giant with one round eye in the middle of its forehead. Cyclops were herdsmen and builders of giant (cyclopean) fortifications. They also worked for Hephaestus at his forge, where they forged Zeus' thunderbolts. It is fortunate that they were only mythical characters.

The **Cyklop** typeface was designed and cast in lead by the "J. Idźkowski i S-ka" Warsaw foundry. It is a very heavy sans-serif two-element typeface, produced only in the oblique form, in sizes from 8 to 48 pt. It was frequently used for newspaper titles and for one-off prints like posters, forms, labels or invitations.

The **Cyklop** typeface was designed in the 1920s by the Warsaw "Idźkowski i S-ka" foundry. This two-element, sans serif typeface is characterized by strong contrast. The vertical stems are far thicker than the horizontal stems. The internal letter openings are in most cases of the shape of an elongated rectangle. This gives the glyphs their unique shape.

Cyklop, in the form of lead type, was produced only in the oblique variant in sizes from 8 to 48 pt. Quite probably, the upright version was not designed and hence not produced. **Cyklop** was very often used for newspaper titles and jobbing prints. Typesetters often reached for it during the whole period between the wars, and continued to use it in the underground newspapers during the second world war. It continued to be used until the beginnings of offset printing

and computer typesetting. Nowadays it is difficult to find it in the form of metal type.

The present font was generated with MetaType1. I extended it to cover the complete set of accented Latin characters and those glyphs which were missing from the original set. I also set out to create the upright variant, which has proved to be a more complicated task than it initially seemed. I hope to be able to release the *beta* version of the **Cyklop** fonts at the end of this year.

The slides of the presentation, cyklop.pdf, are available from the "presentations" folder at <http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007>. To get the current version of the font files, visit the GUST e-foundry site at <http://www.gust.org.pl/projects/e-foundry>.

ABCD
EFGHIJKL̸MNO
P̸QRSTUWXYZ
a̸bc̸c̸de̸f̸gh̸ijk
l̸mno̸o̸p̸q̸rs̸stu
v̸wx̸y̸z̸z̸z̸
0 1 2 3 4 5 6 7 8 9
([, , € \$ % & ! ? \$ ")

Do we need a font system in T_EX?

Hans Hagen
PRAGMA ADE
<http://pragma-ade.com>

Abstract

In this article I will reflect on the font system(s) currently built into ConT_EXt. From this perspective I will mention the current directions in the development of T_EX engines and how they may influence ConT_EXt. I will also put this in the context of document layout design.

1 Introduction

This article was written while Taco Hoekwater and I were working on LuaT_EX and ConT_EXt MkIV, work that is ongoing. This process gives us much time and opportunity to explore new frontiers and reconsider existing ConT_EXt features. We also use this process to write down some ideas as they evolve. Much detail is missing and I assume that the user is somewhat familiar with fonts.

Since this article is not typeset using my regular T_EX setup I will not give many examples. After all, it's just meant as a teaser for users who want to discuss future font support in T_EX (especially in ConT_EXt). In the related presentation I will give a few examples.

2 Starting point

One of the characteristics of a T_EX macro package is that it provides some kind of font system. Such a system deals with two issues:

- consistent switching between different styles and sizes (mostly in text mode)
- handling relative scales of fonts in super- and subscripts in math mode

The T_EXbook and its related plain T_EX format demonstrate what such a system may look like. However, it sets up a 10-point system and when users want, for instance, a 12-point setup, more definitions are needed. As soon as a user switches between 10 and 12 points a whole set of commands needs to be redefined or at least commands need to adapt their behaviour.

3 Into context

The macro package ConT_EXt evolved over time and in principle permits you to set up your own font system, but in practice users will use the built-in font support which is organized as follows.

The main classification is *style*. Examples of font styles are serif (rm), sans (ss) and mono (tt) but math (mm) is also a style. Of course this is a rather

arbitrary classification, but it's kind of rooted in the fact that Computer Modern came in these variants. When I started using the Lucida fonts I introduced handwriting (hw) and calligraphic (cg) styles and more are possible.

Next we have *style alternatives* such as normal, slanted, bold, italic and the like. Again, these are rooted in the fonts that came with T_EX. Slanted is kind of artificial and italic is not always really italic, which is why the term 'oblique' is used as well.

Then comes *size*. In addition to the normal size one can switch between predefined but configurable additional sizes: larger ones denoted by the characters a, b, c, etc., and smaller sizes by x and xx.

Font switches are either written as part of the source stream or set as property of (structural) elements. Examples of stream commands are:

```
\rm \tt \tf \tfx \bf \bfx \bfxx  
\sl \sla \slb \tttfx
```

Style properties are defined and used like this:

```
\setuphead[subsection][style=bold]  
\definefontalternative[LargeAndBold][\bfd]  
\setuphead[section][style=LargeAndBold]
```

If a user is in control of the style, such a system works rather well. One can conveniently switch to a different style, alternative and/or relative size.

4 Typefaces

When one is not in control of the document design, there's always a chance that one has to deal with yet another level of organization. Think of a journal where some articles are typeset with FontFont Meta for the running text combined with Lucida Math, and other articles are typeset in Palatino for both text and math. Add to that yet another choice of fonts for the headers and footers and we're talking of three distinctive font setups for one publication.

This is where typefaces come into play. These are combinations of styles within one collection. One can for instance define a typeface *palatino* which is a combination of Palatino Nova (serif), Palatino Sans

(sans) and Palladio Px (math) completed with Latin Modern Typewriter (mono). Of course we need to make sure that we scale the Latin Modern to match the Palatinos. The following definitions were used for the reader of the ConT_EXt conference in Epen (2007):

```
\definetypface[mainface] [rm] [serif]
  [palatino-nova-regular] [default]
\definetypface[mainface] [ss] [sans]
  [palatino-sans-regular] [default]
\definetypface[mainface] [tt] [mono]
  [latin-modern-light] [default]
```

```
\definetypface[extraface] [rm] [serif]
  [palatino-nova-regular] [default]
\definetypface[extraface] [ss] [sans]
  [palatino-sans-informal] [default]
\definetypface[extraface] [tt] [mono]
  [latin-modern-light] [default]
```

These are applied with:

```
\setupbodyfont[mainface]
\setuplayout[style=
  {\switchtobodyfont[extraface,sans]}]
```

The `default` parameter selects the scaling model, in this case not based on design sizes, but derived from 10-point variants.

To make life (and choosing) even more complex, users more and more run into fonts that come in different weights (light, regular, medium, dark, ultra), thus ending up with multiple typeface definitions becomes the norm. It also means that users will always have to face the difficulties of font definitions: the burden of too much choice. What combination looks best?

```
\starttypescript [mono]
  [latin-modern-regular] [name]
  \usetypescript[mono] [fallback]
  \definefontsynonym[Mono]
    [lmtypewriter10-regular]
  \definefontsynonym[MonoItalic]
    [lmtypewriter10-oblique]
  \definefontsynonym[MonoBold]
    [lmtypewriter10-dark]
  \definefontsynonym[MonoBoldItalic]
    [lmtypewriter10-darkoblique]
\stoptypescript
```

```
\starttypescript [mono]
  [latin-modern-light] [name]
  \usetypescript[mono] [fallback]
  \definefontsynonym[Mono]
    [lmtypewriter10-light]
  \definefontsynonym[MonoItalic]
```

```
[lmtypewriter10-lightoblique]
\definefontsynonym[MonoBold]
  [lmtypewriter10-regular]
\definefontsynonym[MonoBoldItalic]
  [lmtypewriter10-oblique]
\stoptypescript
```

Did I discuss design sizes yet? Computer Modern comes in design sizes. Apart from the esthetic aspect, this made much sense in a time where bitmap fonts were the rule. I must admit that I have no other fonts on my machine that come in design sizes. The core font system of ConT_EXt is set up with design sizes in mind, but later extensions made defining typefaces based on one design size convenient (normally 10 point). For this reason users will never deal with the low level font definition system directly.

Recently we see design sizes come back in another disguise. Instead of variants in terms of size we get ‘caption’ and ‘display’. Technically one can embed different design sizes in an OpenType font but this does not happen often yet.

5 Simple definitions

Occasionally we needed a special font definition, for instance when typesetting a title page. There we can use definitions like

```
\definefont [TitleFont] [SerifBold sa 3.5]
```

This means as much as: define a font *TitleFont* which uses the current *SerifBold* (symbolic names are used all over the place in the definitions, aka typescripts) and scale it to 3.5 times the current `bodyfontsize`. This means that we’re freed of hard coded (and cryptic) font file names.

6 Features

One thing to keep in mind when setting up fonts is the font encoding. An encoding is a subset of glyphs out of the whole repertoire available in a font. Font encodings (not to be confused with file encodings or input regimes) are a side effect of T_EX being an 8-bit system, a restriction which is removed by Omega (Aleph), X_ƒT_EX and LuaT_EX. Other characteristics are mappings (from upper- to lowercase and reverse) and, more recently, features as part of OpenType fonts.

For typesetting the mentioned reader I used LuaT_EX in combination with the experimental ConT_EXt version MkIV and so the OpenType variants could be used. The fact that the font itself provides features puts some demands on the font system. How do we pass them to T_EX (in the case of X_ƒT_EX) or Lua (in the case of LuaT_EX)? In X_ƒT_EX

one can say:

```
\font\MyFont=
  palatinonova-regular:liga;dlig; at 12pt
```

But this does not go well with the abstraction and separation of name and style in ConTeXt. In LuaTeX one can implement any interface but at the price of taking care of translating features defined in the font into something that TeX can deal with. This is a fundamental difference with XeTeX: it takes a macro package writer more effort to provide font support in LuaTeX, but this is compensated by more flexibility. As with XeTeX we expect macro package writers to take care of that.

Recent versions of pdfTeX also introduced features, like *hz* optimization and protruding. These features can be applied to individual fonts as well as to styles and typefaces. For this we can use the ‘font handling’ subsystem that we will not discuss in this article. In short, that subsystem deals not with real font features, but with TeX applying its own features to a font.

In pdfTeX one can add inter-character spacing to a font using the low level commands:

```
\font\MyFont=somefont at 12pt
\letterspacefont
  \MyLetterSpacedFont=\MyFont 50
```

This means that 0.025 em is added on each side of a character. The problem with this TeX feature is that it refers to an already defined font. Also, one has to compensate for spacing before and after the sequence of characters manually. What complicates matters even more is that each feature uses a slightly different interface and that features are applied to global font definitions. Such low level commands are not something the average TeX user wants to deal with so we need some kind of high level interface.

Because the distinction between font features and TeX features is somewhat fuzzy, we will use the term features for both. From the user’s perspective it does not really matter.

7 Interface

For a ConTeXt user, a more natural interface is the following:

```
\definefeature [myfeatures]
  [ligatures=yes,oldstyle=yes,spacing=.025em]
\definefont [MyFont]
  [somefont] [feature=myfeatures]
```

How do we implement these and other features? Fonts can have small caps and oldstyle numerals. One may want these but not always. Here we face a dilemma: do we need a complete small caps typeface (many definitions) or is it just an alternative

selection of glyphs. When we set up the base font system small caps were often of limited availability, so it ended up as an alternative by default. However, now that we have enough memory in our machines, and now that fonts often come with small caps in all styles and alternatives, we can equally well define it as an additional typeface. So, we can define a *palatino* alongside a *palatino-sc* and *palatino-os*.

Consider the regular shapes. In this case a macro package can decide to create three fonts out of, say, PalatinoNova-Regular: a normal one, one with lowercase characters replaced by small caps, and one with digits replaced by oldstyle numerals. But the package can also decide to pass the font as it is to TeX and at some point in the typesetting process swap lowercase characters by uppercase ones, and/or replace digits. This saves two font instances at the cost of some extra processing. Because the design of the document often includes a consistent choice for oldstyle numerals, it makes sense to create the extra font here, but in the case of small caps I’m not sure which alternative is better.

You may wonder what this has to do with interfacing so let’s give another example. Sometimes a large chapter or section head looks better when a bit of inter-character spacing is applied. Do we create a spaced font for just a few occasions or do we move that to internal (node) processing? Defining a truckload of extra fonts just because we want to space a few times does not really make sense. Also, a spaced font is no real solution because one has to deal with the begin and end of a spaced sequence then.

What does a user actually want to tell the system? Is it:

```
some text {\UseMyLetterSpacedFontHere
  some text} some text
```

or maybe:

```
some text {\LetterSpacedThisText
  some text} some text
```

In the first case the user asks for a font switch, but in the second case we’re dealing with a property which is not really related to a font at all, apart from the fact that the spacing may depend on font characteristics. I can also envision several variants: spacing based on character kerning, or equally spaced fonts, or slightly randomly spaced.

Or consider that at some point you want to use the outline variant of a font. This is a drawing property, not so much a font property, so again, the second approach may make more sense.

So, certain features may influence the interface as well: are we talking of a feature attached to a font

definition, or is it applied to a range of characters (glyphs) in the document? In the first case we need to enable the feature when we define the font, but in the second case we can do that in the style when it's needed. What do users prefer most?

8 Frontends

For over 25 years the \TeX engine was essentially frozen. With $\epsilon\text{-}\TeX$, some programming features were added, Omega added directional typesetting and pdf \TeX built in the backend. Speaking for Con \TeX t none of them really demanded a redesign of the macro package as a whole or one of its subsystems. Even X \TeX with its font features could be supported rather easily until the moment that the name specification was extended to support font names as well as filenames at which point the low level interface (using brackets) started interfering badly with the Con \TeX t user interface. The greatest differentiation was in the handling of backends and that was implemented by separating specific backend code into driver files (think of color support or graphic inclusion).

The differences in frontends were negligible and could be dealt with by code branches or selective macro definitions at format generation time. However with the evolution of font systems, the frontend part became more tricky, and not only from the perspective of user interfacing. Suddenly we were dealing with features being present or not, or being implemented differently. So, from now on, even with a consistent user interface the users need to be aware of what exactly is supported by the frontend and with the font itself. We have to see where this leads.

9 Math

We have hardly mentioned math, so how about it? A substantial part of \TeX and therefore its font machinery is dealing with math. Math in \TeX is a family business. A family groups fonts in sizes: normal, small and smaller. Following the Plain \TeX tradition we use families for math roman, italic, symbols, extension symbols as well as what we previously called alternatives (bold and so on).

And there the problem strikes. First our population only counts 16 families, which is not enough to deal with regular, slanted, italic, bold, bold italic, all kinds of extra symbols, also in variants, and more. Another complication is that one may want to use bold text but not bold math or the reverse. Add to this that \TeX is programmed in such a way that changing families mid-formula is not an option (the last definition counts), you can imagine that it's hard to please users in this area. More families

would make life easier, but that only partially balances the equation of demand and supply. Font encodings also may play a role here: specific math encodings and regular text font encodings (not all math documents are written in English).

10 Daily practice

If after this exploration you're still with us, we're ready to review this system. Over the years the Con \TeX t user base has widened and the range of applications is impressive. This also means that we need to provide the current font related subsystems in future versions. Where do we stand with a font system that is set up for consistency and convenient definition of fonts in terms of base characteristics?

My own application of Con \TeX t ranges from special applications, via manuals, to (often) fully automated generation of documents as part of a bigger workflow. For the last group of applications we have to provide the mechanisms as well as the styles. Most of the styles that I have to define are prototyped in desktop publishing applications. Not only is any systematic approach to using fonts missing, also many fonts are mixed together. This means that in practice one can forget about a proper font system. Of course I try to fit these into some kind of system, but since the input is often rather simple too, font usage is also predictable. I frequently end up with a simple typeface definition for the main body font where I also define the math and monospaced variants, because one never knows what fallbacks are needed.

Life is actually worse: designs are seldom consistent in terms of font usage, color application, (interline) spacing, layout and structure. But these are the cornerstones of Con \TeX t and that means that in such cases they are much of what Con \TeX t provides (no big deal because we have hooks all over the place).

11 Open type

At the same time we see OpenType fonts showing up and these provide features that were not available and/or were distributed over multiple fonts. On the one hand, these (often Unicode) base fonts are great, especially when used with a modern \TeX implementation. Quite often I get specifications in a way that indicates that the designer thinks in terms of her/his application. For instance, when 10 pt is specified, in most cases 10 bp (or PostScript points) are meant. And is an 'H-height' the same as an 'X-height'? I'm not sure that the abundance of features in OpenType fonts will be dealt with consistently and with care.

Here is an example: schoolbooks that teach kids French are typeset using systems that come preconfigured for English. Suddenly fonts have language-related features and you can bet that these are used. However, in the past, awareness of such features is dim. How many schoolbooks use the proper French spacing around colons and semi colons? And how many use the right French quotation symbols? If it does not happen today, how about the future? How consistent will designs be? Just watch how suddenly we see those relatively unknown ligatures (like `st`) show up, simply because they are there. The fact that these are language dependent does not bother some users.

As it happens, support for languages in \TeX has always been quite strong. Users are aware of their language needs, and \TeX supports them. Actually, in many areas \TeX provides a lot of detailed control, and this may conflict with less sophisticated control driven by fonts. We cannot assume that all font designers and foundries pay an equal amount of care to each (often big) OpenType font.

The number of available math fonts is not large. This means that when those are converted to OpenType and use the Unicode encoding, we can get rid of many nasty tricks at the macro level. There will be no more need for tricky family magic, nor for font switches at unfortunate moments: we only have a few fonts left. Because $\text{Lua}\TeX$ provides a way to define virtual fonts on the fly, missing bits and pieces can be filled in and style alternatives can be provided even if the math fonts themselves lack them. Of course this only works out well if we are willing to rewrite and/or extend parts of macro packages substantially.

12 Control

So, in addition to the question whether we need a full-blown font system, we need to ask ourselves where we let the font drive the machinery (the font controls \TeX) and where we let \TeX be in control (\TeX controls the font). In $\text{Lua}\TeX$ we (the $\text{Lua}\TeX$ team) provide access to the font definition mechanisms, which permits macro package writers to let the font be the driving force. For instance, one can define a font complete with ligature information and let \TeX do the job. But one can equally well bypass this mechanism and process node lists (one of \TeX 's internal representations of the typeset text) by using special Lua code hooked into \TeX . Or take the mentioned kerning around French punctuation: this can be a font property but also a matter of node processing. Because most \TeX users leave such details to macro packages, one can expect both solutions to

show up. Instead of hard coding alternatives in the \TeX kernel, we just provide the machinery to macro writers.

Recently I had to write a style for a project and rewrite it many times because automated typesetting suddenly forces those involved to pin down designs. It's often hardly a challenge for a \TeX user to identify the inconsistencies between different volumes of a series of books (equally well one can identify systematic problems with \TeX macros because they show up each time). When reverse engineering an existing design inconsistencies creep in, and quality control depends on which volume is taken for comparison today. In this case it also happened that the design of this series was based on a font that was not only very incomplete, but also buggy. Familiar characters were missing, names in the encoding vector were wrongly applied. So, we had to come up with a special font encoding that in itself was wrong with regards to the names used. This is a bit of a nightmare because a different encoding results in extra map files as well an extra instance of hyphenation patterns, i.e. another format file.

In $\text{Lua}\TeX$ this can be done differently. There one can add some code to the loader that takes care of special remapping and filling in gaps with placeholders. Of course this can be embedded in a higher level user interface. I already have quite a lot of experimental code marked to be turned into production code some day.

13 Conclusion

When dealing with complex and/or very structured documents we can benefit from a font system as currently found in $\text{Con}\TeX$ t. Users can be sure that when they switch to another style or alternative, that the system will follow.

But what about a font system for situations where \TeX has to compete with (or replace) desktop publishing? There we can roughly conclude the following.

- We can stick to a simple font model: one size for the main text, a few definitions for different alternatives (regular, bold, italic, bolditalic) because this is what the designer has available.
- In addition we have to define a truckload of fonts for all kind of elements (structure, ornaments, bits and pieces of the page body, captions, tables, etc.). We can stick to dumb font switches since the (structural) editing tools used don't permit anything beyond the specs anyway.
- Small caps, oldstyle numerals, inter-character spacing in titling, and so on can be applied

when constructing an internal font representation or handling can be delayed to node processing time. Depending on the quality of the font, some tweaking needs to be done. It is still open whether we treat them as font features or as a property of a part of the text.

- Math is a different story. If dealing with third party input, it's often more a matter of cleaning up than of advanced font trickery. Unicode math fonts may simplify our life but may as well complicate it. But whatever solution we end up with, more families are welcome. We need to be prepared for exceptions (especially when dealing with specialized math, schoolbook math) and also need to keep in mind that T_EX no longer dominates this market or at least is fed with input coming from word processors with math editing capabilities.
- Advanced features like hz and protruding are of course possible but will often be interpreted as errors by QA people. Applying them in T_EX is not complex, but explaining them to designers may be. Anyway, in most cases ragged right is to be used, if only because designers don't trust systems to do a proper justification. Here T_EX's 25 year reputation of creating nice paragraphs does not help much.

It goes without saying that a simple font system will be faster than an advanced one normally used in T_EX. So, any time that we lose in processing node lists, we may well gain back in a simplified font system. On the other hand, life may become more complex now that T_EX engines provide more (distinctive) font related features, which in turn may drive user demand into all directions possible: *I want these ligatures but not those!* This may be compensated for by the fact that we need to load fewer fonts, and get rid of font encodings and character/glyph fall-back trickery.

In retrospect, the way the plain T_EX format defines fonts is not that bad for most situations where some third party is responsible for the overall document design. The complication arises when one writes manuals and needs to switch frequently between sizes and styles.

Actually many dirty tricks used in macro packages also result from the simple fact that one needs to typeset user manuals about T_EX, which means that one has to deal with characters in special ways which in turn may be reflected on the font system.

Of one thing there can be no doubt: the landscape of font usage is changing and T_EX macro packages have to adapt.

OpenType fonts in LuaTeX

Taco Hoekwater
<http://luatex.org>

Abstract

Since the start of February 2007, LuaTeX has supported the use of OpenType fonts directly, without the need for separate metrics and font map files. This talk will explain and demonstrate how this works in practice.

1 Introduction

This is an updated version of the paper that was in the preprint. There has been considerable progress in the time between the preprint (early April) and now (early August). The current paper documents the state of affairs in LuaTeX 0.10, the first public beta.

2 OpenType fonts in LuaTeX

If you want to do typesetting with TeX, you have to get the required font metric information from somewhere. METAFONT- or fontinst-based fonts typically come as a set of TFM and VF files, and for those, LuaTeX behaves in a way that is backward compatible with any other traditional version of TeX.

But loading metrics for OpenType (`.otf`) and TrueType (`.ttf` and `.ttc`) fonts can also be done through a Lua extension interface, in which case there is no need for TFM, VF, ENC, and MAP files. Instead, you (or more precisely, a macro package writer) has to write a bit of Lua code.

There are two separate parts to this process, that will be explained in turn in the next paragraphs:

1. You have to make LuaTeX use the Lua extension interface instead of the compatibility mode metrics loading, by setting up a Lua callback.
2. You have to write the necessary Lua code to make it possible for LuaTeX to use the OpenType fonts you have installed.

3 Font definitions through Lua callbacks

Installing ‘callbacks’ is one of the most important concepts in LuaTeX. A callback is what we call the situation whereby LuaTeX is instructed to run a user-supplied Lua function instead of a bit of internal compiled code. A few dozen of these interception points are defined at this time, and they have all been given names.

You install a callback by connecting a Lua function to one of these names. For this purpose, there is a predefined ‘register’ Lua function provided. The

most relevant callback for font definitions is named ‘define_font’, and it could be set up like so:

```
\directlua0 {  
  function read_font (name, size, fontid)  
    local file = kpse.find_file (name, 'tfm')  
    local metrics = font.read_tfm (file, size)  
    return metrics  
  end  
  
  callback.register('define_font', read_font)  
}
```

This example first defines a function to do the work (`read_font`), and then registers that function as a callback. The function does essentially the same as what TeX would have done without any callback. It uses the functions `kpse.find_file` and `font.read_tfm`, which are predefined helper functions.

When LuaTeX next runs into a `\font` command, it will gather the user-supplied font name and size specification, and pass those values on to the Lua function `read_font` as the first two arguments. The task of `read_font` is to create a data structure (in Lua this is called a ‘table’) that contains the metric information needed for typesetting in the font `name` loaded at size `size`.

The internal structure of the Lua table that is to be returned by `read_font` is explained in detail in the LuaTeX manual. Fortunately for the length of the example, that structure is a super-set of the structure returned by `font.read_tfm`, so it can just be passed along without further manipulation.

In the example you can see that there is a third argument to `read_font`, ignored in this case. LuaTeX also passes the internal id number of the font that is going to be defined. This is because, in macro packages, it is not abnormal for the same font to be defined more than once using the same `name` and `size` specification, so instead of returning a Lua table defining the metrics for a font, it is also legal to return just a number, referencing the `fontid` of an already defined font. That way, you could set up a lookup table of already defined fonts.

4 Handling OpenType fonts

There is a Lua module included in LuaTeX that can be used to read a font's metrics from the disk, using the font reading code from the open source program FontForge.

The contents of this module are available in the Lua table named `fontforge`. Using it, the basic way to get the metric information is like this:

```
function load_font (filename)
  local metrics = nil
  local font = fontforge.open(filename)
  if font then
    metrics = fontforge.to_table(font)
  end
  return metrics
end
```

This code first loads the font into program memory with `fontforge.open`, and then converts it to a Lua table by calling `fontforge.to_table`.

The font file is parsed and partially interpreted by the font loading routines from FontForge. The file format can actually be any one of OpenType, TrueType, TrueType Collection, CFF, or Type 1.

There are a few important advantages to using this approach with a dedicated Lua module, compared to having a single dedicated helper function to read an OpenType font file:

- The internal font encoding is automatically processed, so that the returned `metrics` table also contains the Unicode encoding information for all the included glyphs.
- Many OpenType features are pre-processed into a format that is easier to handle than just the bare feature tables would be.
- And looking at it from the other side: it is still possible to completely alter any feature you want to change, as nothing at all is hardwired in the executable.
- PostScript-based fonts do not store the character height and depth in the font file (in Type 1 fonts, this information is in the AFM file, in CFF fonts it is not present at all). For CFF fonts, this means that the character bounding box has to be properly calculated, a task that is handled internally by FontForge.

- In the future, it may be interesting to allow Lua scripts access to the actual font program, perhaps even creating or changing a font file itself.

However, there is also a downside: the data structure of the table returned by the OpenType reading routines is very low-level and very close to the internal format used by FontForge itself.

This means that it is not compatible with the table structure required by the font definition callback, so some modifications to the structure are needed before the table can be passed on to LuaTeX proper. This area is still under development. We plan to provide a set of helper functions for this task eventually but for the moment, this has to be done by Lua code you write yourself.

To finish off this introduction, here is a small peek into the table returned by `fontforge.open`. What follows is a human-readable representation of the ligature glyph for ‘fi’ in the font `lmroman10-regular.otf`:

```
{
  ["boundingbox"]={ 27, 0, 527, 705 },
  ["lookups"]={
    ["ls_l_10_s"]={
      {
        ["specification"]={
          ["char"]="f_i",
          ["components"]="f i",
        },
        ["type"]="ligature",
      },
    },
  },
  ["name"]="f_i",
  ["unicodeenc"]=64257,
  ["width"]=556,
}
```

Font-specific issues in pdfTeX

Hàn Thê Thành
River Valley Technologies
<http://river-valley.com>

Abstract

In this paper I try to give a summary of some font-related topics in pdfTeX. Some of them are already described in the pdfTeX manual, such as font expansion and margin kerning, some have been mentioned only in various places like relevant mailing lists, README or example files coming with patches, and email exchanged between people interested in a particular topic. This article attempts to put everything into one place, hoping to make it easier to follow.

1 Introduction

A large part of the pdfTeX extensions is related to font handling. Having an overview of all those font-related issues is not always easy, since the pdfTeX manual is a somewhat dry thing to read as a whole. Apart from that, there are also things that are not described in the manual yet. In this paper I will try to give an overview of font extensions in pdfTeX. Instead of listing all relevant primitives with their description, I will write on particular topics that I consider interesting to mention here.

2 Font expansion and margin kerning

Since these features have been mentioned many times, I simply skip their description here and only give the references to relevant sources: [1], [2].

L^AT_EX users who want to try out these features should start with the L^AT_EX `microtype` package. ConT_EXt users should consult the ConT_EXt manual first.

Primitives relevant to font expansion:

- `\pdfadjustspacing`,
- `\pdffontexpand`,
- `\efcode`;

Primitives relevant to margin kerning:

- `\pdfprotrudechars`,
- `\lpcode`,
- `\rpcode`.

All those primitives are described very well in the pdfTeX manual.

3 Additional micro-typographic features

Apart from the above features, pdfTeX has some additional support for finer control on interword spacing and kerning. The `microtype` package provides an easy access to those features. Furthermore,

the `microtype` manual [1] has a very good introduction to these additional features, which I copy here for convenience (slightly edited):

... On the contrary, pdfTeX was extended with even more features: version 1.30 introduced the possibility to *disable all ligatures*, version 1.40 a robust *letterspacing* command, the *adjustment of interword spacing* and the possibility to specify *additional character kerning*.

Robust and hyphenatable *letterspacing* (tracking) has always been extremely difficult to achieve in T_EX. Although the `soul` package undertook great efforts in making this possible, it could still fail in certain circumstances; even to adjust the tracking of a font throughout the document remained impossible. Employing pdfTeX's new extension, this no longer poses a problem. The `microtype` package provides the possibility to change the tracking of customisable sets of fonts, e.g. small capitals. It also introduces two new commands `\textls` and `\lststyle` for ad-hoc *letterspacing*, which can be used like the normal text commands.

Adjustment of interword spacing is based on the idea that in order to achieve a uniform greyness of the text, the space between words should also depend on the surrounding characters. For example, if a word ends with an 'r', the following space should be a tiny bit smaller than that following, say, an 'm'. You can think of this concept as an extension to T_EX's 'space factors'. However, while space factors will influence all three parameters of interword space (or glue) by the same amount — the kerning, the maximum amount that the space may be stretched and the maximum amount that it may be shrunk — pdfTeX provides the possibility to modify these parameters independently from one another. Furthermore, the values may be set differently for each font. And, probably most importantly, the parameters may not only be increased but also

decreased. This feature may enhance the appearance of paragraphs even more. Emphasis in the last sentence is on the word ‘may’: this extension is still highly experimental — in particular, only ending characters will currently have an influence on the interword space. Also, the settings that are shipped with `microtype` are but a first approximation, and I would welcome corrections and improvements very much. I suggest reading the reasoning behind the settings in section 15.8.

Setting *additional kerning* for characters of a font is especially useful for languages whose typographical tradition requires certain characters to be separated by a space. For example, it is customary in French typography to add a small space before question mark, exclamation mark and semi-colon, and a bigger space before the colon and the guillemets. Until now, this could only be achieved by making these characters active (for example by the `babel` package), which may not always be a robust solution. In contrast to the standard kerning that is built into the fonts (which will of course apply as usual), this additional kerning is based on single characters, not on character pairs.

The possibility, finally, to *disable all ligatures* of a font may be useful for typewriter fonts.

The `microtype` package provides an interface to all these micro-typographic extensions. All micro-typographic aspects may be customised to your taste and needs in a straight-forward manner.

3.1 Letterspacing

We all probably know what letterspacing is and related problems when using it with TeX. The robust and reliable way to letterspace a font in TeX is to create a virtual font which inserts a fixed kern around each character. The famous `fontinst` package can be used to do this, however, it must be done for each font we want to letterspace. Furthermore, `fontinst` is not a tool for everybody.

There have been several attempts in pdfTeX to solve this problem: one idea was to insert an explicit kern before and after each character, when the character is typeset by TeX, roughly like typing i. e. `\kern.1em X\kern.1em` for each character ‘X’. This approach had several problems; the most serious one is that it disabled hyphenation. Another attempt used implicit kerns instead of explicit ones; while this method allowed hyphenation, it caused other problems. In the end, a method that generates a virtual font on-the-fly was implemented. It works very much like the way one uses `fontinst` to letterspace a font, but it is done automatically in pdfTeX, at run time. A minimal example looks like this:

```
\font\f=cmr10
\letterspacefont\fx=\f 100
\fx <letterspaced text>
```

The above commands create a letterspaced version of `\f` (which is `cmr10`) as a virtual font. This virtual font is accessible to the user via the control sequence `\fx`. Each character from `\fx` is typeset using its counterpart from `\f`, plus a kern of $50*\quad(\f)/1000$ at each side.

There are some issues with compensating for the kern at the beginning/end of letterspaced text. Since the kern amount is known, it is possible to compensate that kern manually if needed, for example when using `\fx` inside an `hbox`.

In a multiple-line paragraph, one can compensate for the kern at the margin using margin kerning like follows:

```
\pdfprotrudechars=2
\newcount\n
\n=0
\loop
  \lrcode\fx\n 50
  \rprcode\fx\n 50
  \advance\n 1
\ifnum\n<256\repeat
```

This is still not perfect, since you lose the effect of margin kerning (now all marginal kerns are the same, so the margins are aligned mechanically as in the case without margin kerning). If you want to have both letterspacing and margin kerning, you need to compensate for the margin kern as follows, given that you have set up margin kerning for `\f` already:

```
\newcount\n
\newcount\m
\n=0
\loop
  \m=\lrcode\f\n
  \advance\m 50
  \lrcode\fx\n \m
  %
  \m=\rprcode\f\n
  \advance\m 50
  \rprcode\fx\n \m
  %
  \advance\n 1
\ifnum\n<256\repeat
```

The current version of pdfTeX (1.40.3) still has a problem when using letterspacing with font expansion. This problem will be fixed soon (not hard to do).

Relevant primitive: `\letterspacefont`

3.2 Adjustment of interword spacing

TeX treats all interword spaces from input text as glue items, while sometimes people need finer control over interword spaces, since this is one of the most important elements in paragraph building. Instead of describing the topic using my own words, I find it more convenient to quote the conversation via email between me and people interested in this topic (Frank Mittelbach and Ulrich Dirr).

Frank: what TeX is missing is a way to kern with the white space between words. The Adobe fonts and others might have such kerns but they have been written for software which does use “space chars” not glue.

Thành: I also would like to see the space character to be handled in a different way than it is now. Turning it into glue is probably not the best solution. It disallows fine adjustment of interword spaces to make them optically even rather than mechanically. Kerning with respect to the space can be used to improve this, but it is certainly not sufficient. Moreover, the boundary char mechanism has its limitations.

Frank: It would be a very radical step if one would introduce real space characters which (perhaps) just before typesetting are replaced by glue not early on. But again, we have now stayed and worked with TeX as it is for 20 years and if certain areas and their underlying ideas prove to be insufficient, why not experiment with alternatives?

However one other comment, if you look at what some typographers write about making the white space visually even, it make me wonder if you really can do much good about having “kerns” if you then end up with

```
<last char><kern><interword glue><kern>
<first char next word>
```

i. e. with the middle part stretching or shrinking at a constant rate, or if you really need <glue> adjustments here.

At least this is what some typographers claim: that if you need to shrink the interword glue that this should not be a constant factor as done with TeX but rather differing depending on the letter shapes at each side of the interword space.

Ulrich: I have had a short conversation with Frank (Mittelbach) about an extension/improvement of the paragraph building algorithm. First I thought it would be maybe possible with the help of `\sfcode` or `\spaceskip` etc. but this will not really work.

The idea is — analogous to the tables for expansion and protrusion — to have tables for optical reduction/expansion of spaces in dependence of the actual character so that the distance between words is optically equal.

When reducing distances the (weighting) order is:

- after commas
- in front of capitals which optically have more room on their left side, e.g., ‘A’, ‘J’, ‘T’, ‘V’, ‘W’, and ‘Y’
- in front of capitals which have circle/oval shapes on their left side, e.g., ‘C’, ‘G’, ‘O’, and ‘Q’
- after ‘r’ (because of the bigger optical room on the righthand side)
- before or after lowercase characters with ascenders
- before or after lowercase characters with x-height plus descender with additional optical space, e.g., ‘v’, or ‘w’
- before or after lowercase characters with x-height plus descender without additional optical space
- after colon and semicolon
- after punctuation which ends a sentence, e.g. period, exclamation mark, question mark

The order has to be reversed when enlarging is needed.

Note: The principle of how this works can be seen in figure 1, where the numbers indicate the preference/order of each interword space when it needs to be stretched/shrunk.

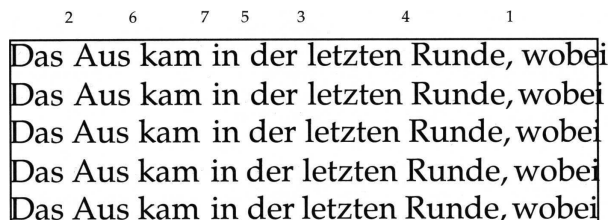


Figure 1: Interword spaces should be changed with respect to the adjacent characters.

Thành: I remember discussing this issue long time ago, when Frank also got involved. The problem with interword spaces in TeX is that Knuth decided to treat interword spaces like glue, while IMHO it needs special care because this is one of the most important factors in building a paragraph and hence we need a way to distinguish it from other glues.

From the experience with margin kerning, I think we should better make some small steps to see whether it makes sense, rather than start heavily changing the paragraph building engine.

Thành: I implemented an approach to allow more control on interword space as we discussed before. Sorry for the long delay.

I introduced three primitives:
`\knbscode` — kern before space code,

`\stbscode`—stretch before space code,
`\shbscode`—shrink before space code.

These primitives have the same syntax as `\rprcode` etc., i. e.

```
\knsbcode\font'\.=200
```

means that if a period sits before an interword space (glue), then the interword glue will be increased by an amount of $1\text{em} \cdot 200/1000$, i. e. the value is given in thousandths of an em as in the case of `\rprcode` etc. `\stbscode` and `\shbscode` are similar but adjust the stretch/shrink components of the interword glue.

Adjusting the interword glue only has effect when the space factor of the previous char is different from 1000.

For now I leave out ligatures and the case *after* the interword glue.

A minimal test file might look as follows:

```
\font\font=cmr10
\pdfadjustinterwordglue=1
\sfcodes'\.=1000
\knsbcode\font'\.=100
\shbscode\font'\.=200
\stbscode\font'\.=300
\font <text>
\bye
```

The above example would adjust every interword glue following a period by adding an amount of $.1\text{em}$, $.2\text{em}$ and $.3\text{em}$ to the glue width resp. shrink resp. stretch component. The primitive `\pdfadjustinterwordglue` is to switch the feature on/off at the global level, and setting `\sfcodes'` to 1000 is required to activate this feature (so they do not interfere with each other).

These features are available in pdfTeX since version 1.40, and are also supported by the `microtype` package. However, the predefined values are not yet optimal—probably more experimenting is needed to tune the parameters to get a good result. Please refer to the `microtype` documentation for further details.

There is no support to adjust the interword space with respect to the next character. The main reason is that it is not easy to do in current pdfTeX code. Hopefully when LuaTeX is ready, this can be changed.

Relevant primitives:

- `\knsbcode`—“kern before space” code,
- `\stbscode`—“stretch before space” code,
- `\shbscode`—“shrink before space” code,
- `\pdfadjustinterwordglue`—turns on/off the feature.

4 Additional kerning

This is a feature that allows inserting a kern before

or after a certain character from a font. A minimal example looks like this:

```
\font\font=cmr10
\pdfprependkern=1
\knbccode\font'\.=500
\font <text>
```

The above example prepends a kern of $.5\text{em}$ before each colon. It is also possible to append a kern after a character:

```
\font\font=cmr10
\pdfappendkern=1
\knbccode\font'\;=100
\font <text>
```

These features are also supported by `microtype` already. However these new features are not flexible enough to get rid of the need to have active characters in `babel/french`, as shown in this email by Daniel Flipo:

I have heard about new kerning facilities coming with pdfTeX 1.40 and started playing with them (through the `microtype` interface, latest version 1.37 2006-09-09 with `\betatrue`). I would love to get rid of the four active characters (`;!?`) in `babel/frenchb`.

Unfortunately, after discussing with Robert (in copy), it appears that these new kerning facilities do not quite fulfil what would be needed for French. I'd like to make a summary of the required specifications in case you can think of a possible solution for future developments of pdfTeX.

1) People who type correctly in French, are used to type a (normal) space before `;!?`. pdfTeX 1.40 can add a kern before them, but cannot do an `\unskip` to remove the typed space. It is hopeless to try to convince French writers to change their habits and refrain from entering a space before `;!?` ; -)

`frenchb` currently handles the four (active) punctuation chars `;!?` in two different ways:

— with the option `\NoAutoSpaceBeforeFDP`, `frenchb` replaces the normal space with an unbreakable one of the correct width, *if and only if* a space (normal or `'`) is present before `;!?`. If no space is typed, `frenchb` does nothing and lets the punctuation mark stick to the preceding word. This avoids a spurious space in URLs (`http://...`), Windows paths (`C:/path`), etc.

— with the option `\AutoSpaceBeforeFDP` (the default), you can carelessly type any of `bonjour!`, `bonjour !` or even `bonjour~!`; `frenchb` will always output it correctly— but then you cannot complain if you get a spurious space in URLs.

2) Another (minor) issue occurs with ‘:’. Again, there are currently two different options in frenchb:

— Most people agree with our « Imprimerie nationale » that ‘:’ should be surrounded by two spaces of the same length, the first one being unbreakable, while the other three (;!?) get a thin space (kern in \TeX) before and a normal space (glue) after. That’s what frenchb does by default.

— Some typographers argue that ‘:’ should be treated like the other three, so an option is provided in frenchb to satisfy them.

AFAIK pdf \TeX 1.40 can add a kern before a character but not a glue, so the spaces around ‘:’ might look asymmetrical in the first case if \TeX stretches the second one.

3) Guillemets are less problematic because they are currently entered with commands ($\backslash\text{og}$ and $\backslash\text{cg}$), not as characters. Spaces after the opening ‘«’ and before the closing ‘»’ should be unbreakable but stretchable (currently these are $.8\backslash\text{fontdimen2 plus } .3\backslash\text{fontdimen3 minus } .8\backslash\text{fontdimen4}$). Moreover, a kern after ‘«’ breaks hyphenation of the following word as Robert already pointed out on the pdf \TeX bug list.

So the current situation still needs improvement, which is likely left to Lua \TeX .

Relevant primitives:

- $\backslash\text{knbccode}$ — “kern before character” code,
- $\backslash\text{pdfprependkern}$ — toggle prepending of kerns,
- $\backslash\text{knaccode}$ — “kern after character” code,
- $\backslash\text{pdfappendkern}$ — toggle appending of kerns.

5 Support for ToUnicode map

ToUnicode map is a concept in the PDF specification that allows mapping from character codes in a font to corresponding Unicode numbers. The purpose is to allow PDF browsers to perform properly operations related to text contents like search, cut and paste. Support for this feature was added mainly to make PDF files produced with the MinionPro package [3] searchable. If you are having trouble with PDF produced by pdf \TeX not being searchable with some fonts, give this feature a try (N.B.: this feature only works for Type1 fonts). A minimal example:

```
\input glyphtounicode.tex
\pdfgentounicode=1
<text>
```

If glyphtounicode.tex is not available in your \TeX distribution, it can be downloaded from [4]. This file covers most common cases. If you want to add your own entries, here is an example how it can be done:

Suppose that you have a font which has another variant of letter ‘A’, named e.g. ‘myCoolA’, and

you wish that glyph to be found when you search for ‘A’. Then you add to glyphtounicode.tex (or insert somewhere in your \TeX file) a line saying:

```
\pdfglyphtounicode{myCoolA}{0041}
```

which means that the glyph with name ‘myCoolA’ has the corresponding Unicode number 0041 (which is the same as for the normal ‘A’). This would make your ‘myCoolA’ behave like ‘A’ regarding operations like search, cut and paste.

6 Support for subfont

\TeX was designed to work with 8-bit fonts only. CJK languages however use fonts with thousands of glyphs. To make those fonts work with \TeX , a trick called ‘subfont’ was developed by Werner Lemberg for his CJK package. The subfont technique splits a huge font into smaller fonts, each of them containing up to 256 characters.

Explaining the subfont mechanism is out of scope for this paper, so I simply refer people with further interest in this topic to [5]. Here I try to give a simple example.

Suppose we want to use the Bitstream Cyberbit Unicode font. This font has about 30 000 glyphs and covers many languages. The fontfile is called cyberbit.ttf. We want to use this font to typeset CJK languages written in UTF-8 encoding.

The first step is to generate the TFM subfonts:

```
ttf2tfm cyberbit.ttf cyberb@Unicode.sfd@
```

The ttf2tfm program is part of the FreeType 1 bundle; it comes with all major \TeX distributions like \TeX Live or MiK \TeX . Unicode.sfd is a *subfont definition* that comes with the CJK package. It is basically a text file containing instructions how to split a large font into subfonts. The above command will produce a bunch of TFM files with names in form cyberbxx.tfm, where xx are two lowercase hexadecimal digits. Copy the TFM files to a location where pdf \TeX can find them.

The next step is to tell pdf \TeX about the subfonts by adding to your \TeX file a line saying:

```
\pdfmapline{+cyberb@Unicode@ <cyberbit.ttf}
```

The effect of the above command is that pdf \TeX will be able to pick up the right glyphs for those TFM files from cyberbit.ttf and embed them as subsetted TrueType fonts in the PDF output. So it is no longer necessary to convert cyberbit.ttf to Type1 subfonts and embed them, or to run ttf2pk. A complete minimal example:

```
\documentclass{article}
\usepackage{CJK}
\pdfmapline{+cyberb@Unicode@ <cyberbit.ttf}
```

```

\DeclareFontFamily{C70}{cyberbit}
  {\hyphenchar\font -1}
\DeclareFontShape{C70}{cyberbit}{m}{n}
  {<-> CJK * cyberb}{}

\begin{document}
\begin{CJK}{UTF8}{cyberbit}
\CJKnospace
<some CJK text in UTF-8 encoding>
\end{CJK}
\end{document}

```

There are many details that are not mentioned here, however the above example should give a good feeling about what can be done.

7 runpdfTeX — a wrapper to run pdfTeX

This section is not about a font-related topic in pdfTeX, but it is also relevant to pdfTeX so I take this opportunity to mention it.

`runpdfTeX` is a wrapper that allows applications to call pdfTeX via a well-defined API in C. The main intention is to hide TeX-specific details from the application that calls pdfTeX to generate a PDF file. A developer can call pdfTeX to convert a TeX file to PDF using library calls that are robust, easy to understand and use, and take care of error handling. This way, a Web developer who is not a TeX expert can set up a system that uses pdfTeX to create PDF output on-demand, for example some report, form, timetable or bank statement. The Web developer can ask or hire a TeX guru to write a TeX file or template that produces the required output.

This is still an experimental project, however I hope it will make pdfTeX more friendly to Web developers who need to create PDF on-demand but are too scared by TeX's complexity to give it a try. The API is available only for C at the moment, but support for other languages will be added. This wrapper has been designed with pdfTeX in mind, but can be used to run other TeX variants as well, for example LuaTeX when it is ready. For further information about `runpdfTeX` see [6].

References

- [1] The manual of the `LATeX microtype` package by Robert Schlicht is available at <http://ctan.org/tex-archive/macros/latex/contrib/microtype/microtype.pdf>
- [2] Hàn Thế Thành, *Micro-typographic extensions of pdfTeX in practice*, *TUGboat*, vol. 25 (2004), no. 1 — Proceedings of the Practical TeX 2004 Conference, pp. 35–38. (Online at <http://www.tug.org/TUGboat/Articles/tb25-1/thanh.pdf>)
- [3] The MinionPro package containing `LATeX` support for Adobe MinionPro fonts is available at <http://www.ctan.org/tex-archive/fonts/minionpro>
- [4] Definitions for ToUnicode entries can be downloaded from <http://pdfTeX.sarovar.org/misc/glyphtounicode.zip>
- [5] The CJK package for `LATeX` is available at <http://cjk.ffii.org/>
- [6] The `runpdfTeX` wrapper is available at <http://runpdfTeX.sarovar.org/>

Those obscure accents...

Karel Horák

Institute of Mathematics, Academy of Sciences, Praha
horakk (at) math dot cas dot cz

Abstract

»A special shape of a háček, similar to an apostrophe, is used in Czech and Slovak with ě, ě, ě and ě characters. It could be derived from the apostrophe or comma, but it should be more humble, smaller, and, importantly, narrower. Generally, the symbol should draw less attention than the comma. This special form could also take a straight shape similar to acute; this usually occupies less space than an apostrophe-like form and it does not cause as many problems in kerning. Vertically, the symbol is most often placed towards the ascender line, but its position does not necessarily have to be constant (with ě, it is often necessary to place the accent higher than with the other characters). With capital ě, it is desirable that the accent exceeds the height of the character. This is mostly equivalent with justifying the upper edge of the accent to the ascender line.«

[DIACRITICS, a project by typo.cz and designiq.cz]

An excursion into history with many examples of good, bad and ugly solutions.

Briefly from the history

The motto quoted in the abstract, which states in the condensed form the final lesson I learned during the long (never finished) way to understand typographic quality, would be sufficient. Nevertheless I try to give a brief summary of this rather old historical problem which (as far as I know) was never touched in our Grand Wizard's creativity (at least not in the five books of the *Computer & Typesetting* series, I suspect).

It is believed that usage of accents in Czech orthography comes from Jan Hus (sometimes quoted as John Huss in English literature) or from his circle (the famous tractate *De orthographia Bohemica* which is also usually designated to Jan Hus originated around 1406). It solves the problem of how to write consonants which do not exist in the Latin alphabet: not as a pair of letters (digraph) but with only one letter, as close as possible to the original one, with some diacritic sign (caron or háček for soft consonants and acute for long consonants).

The accent for softening appears often as a dot, then a hook, not exactly in the same shape as today — in many cases it is closer to the shape of the hook of letters ě or ě:

**Wizteż gatk gest to Dite B
rofklo/ Krystus, duchem geho**

Note that the original form of dot still exists in Polish ż and also that the original digraphs sz and cz are still in use there.

It should be noticed that black letters (fraktur) were widely used in those times for typesetting. And for many years, types were often not created in the country but brought from abroad. Black letters were used in printing until the end of 18th century. Roman letters were used rarely, mostly in titles only. With the exception of Comenius' book published 1659 in Amsterdam the first Czech book printed in roman letters appears only in 1738.

One exception is shown in Comenius' manuscript of *Didactica* (Lešno 1627–32):

DI DACTICA

to gest

Vmenj vmeleho rohviovanj.

Abvachy tovj slovij

Vrijno nej na tele rovrofte aftar smig jaine,
roffemu to coj tu fobike aovdobim pvtomne gubade,
vjfo jivota vjivalej, ftafne, fudne, plne, voj,
vuen a tat poteffene k jivotu obvoj
naktrogen bjt rofl.

Loj feroffe

Micne, zablady z famiche pvi rovej rovatj; probovjce,
v fiamicne, pvtblady pvtijf icme fvojf vovij, voj fvojtage;
votunale, na lera, mifpice voj a fvojij, rozmiejce;
z kromffemu tomu aby kigle pvtimedeno bfo, v
pvtovjem icij, v rade davana.

The change from blackletters to old good roman types came in Bohemia only with a national revival. Roman types served as a symbol of liberation from the German influence. But the problem with appropriate accents survives, as one of the first primers (spelling-books) clearly shows.

^{34.} **á** **š**
ha-de! há-dě, ha-dy.
ha-di, há-ďa-ta, vi-di.
ho-dy, ho-dí, cho-di.
ku-ře đo-bá, dí-ra,
dě-ra-vý, ho-di-ny.
^{35.} **ň** **ř**
hu-ňa-tá ha-le-na.
ba-ňa-tá ná-do-ba.
ko-ře-ní, ku-chy-ně.
hú-ně, bá-ně, ku-ře
bo-jí se lu-ňá-ka.

The name caron seems to be more popular in typography, while háček is widely used in linguistics. In Czech, háček means the diminutive form of ‘little hook’. On the other hand, the etymology of the word caron is quite unclear (there is an idea that it may derive from a fusion of the words caret and macron). In various languages which use this accent, caron is called ‘softener’ or ‘palatalization mark’, ‘little roof’ or even ‘hat’ (in Finnish).

Today the caron is also used by the Slovaks, Slovenians, Croats, Bosnians; Serbs and Macedonians (when romanizing the official Cyrillic); Upper Lusatian and Lower Lusatian Sorbs, Lithuanians, Latvians, and Belarusians (formerly in the Latin alphabet, now only in romanization of the official Cyrillic).

Writing and printing carons

In printed text, the caron combined with letters t, d, l, and L is reduced to a small stroke. This usually does not happen in handwritten text. Although the stroke can look similar to an apostrophe, there should be a significant difference in kerning.

Using apostrophe in place of a caron looks quite ugly though it can be still found in many contemporary examples, such as:

št'áva krat'ákoví

On the other hand, in some cases apostrophe is used as sign of ‘palatization’ in transliteration of some languages, e.g. Russian.

Virtual fonts. Thanks to our Grand Wizard, virtual fonts provide an easy method to prepare a font with the necessary accents, with of course the palatal hook serving as an exception confirming the rule . . . Unfortunately most “localized” versions of commercial fonts distributed not very long ago were prepared in much the same way: i.e. l as l with apostrophe being ‘close enough’. One can see the difference with the text set in metal type

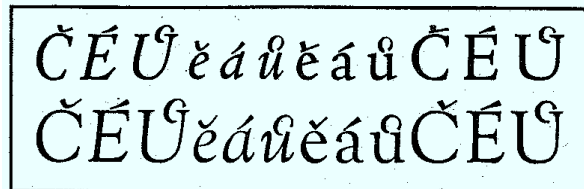
„Přešlo to všecko a teď tu stojíš jako smeták zapomenutý v koutě, když jím po plesu zametli. Anér agathos einai . . . to je přece z Xenofontovy Anabase. Jak stárneme, jak zapomínáme!“

and one of the first electronic versions of Baskerville
 1234567890'!?(.,- —)'«»%&§;
 áéíóúýčďěĺňřšťž
 ÁÉÍÓŔŮŮÝČĎĚĹŇŘŠŤŽ
 äöüÄÖÛàøØæÆåÅßô

The quality of system fonts in Windows is not much better:

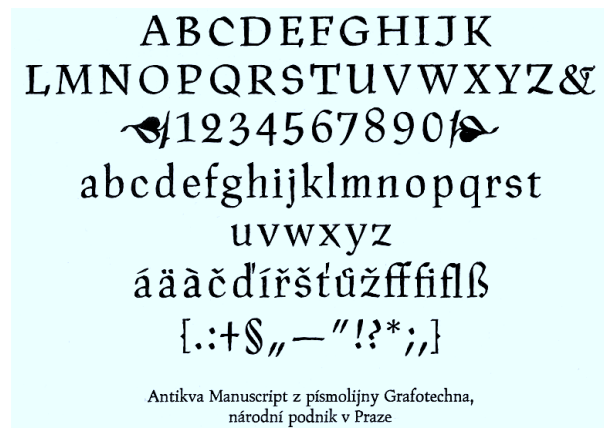
žlut'oučký kůň úpěl d'ábelské ódy!
žlut'oučký kůň úpěl d'ábelské ódy!
 žlut'oučký kůň úpěl d'ábelské ódy!
žlut'oučký kůň úpěl d'ábelské ódy!

(all lines were printed to a PostScript file from the Windows font viewer). Not understanding the text one could guess that there are more than five words in the line! Perhaps all the effort of the best Czech typographers including Vojtěch Preissig, Karel Dyrynk and Oldřich Menhart was not forgotten?



AABCČDĎEĚĚFGHIIJKLMNN
 OOPQRŘSŠTTŮŮVWXYÝŽŽ
 aábcčdďeěfghiijklmňoópqrř
 sšttuúúvwxyýžž 1234567890

The above first example of a Goudy font with Preisig accents may look little bit extravagant, mainly the rather original ring of uppercase U which was eventually not generally accepted, but in a more moderate form can be also found in Menhart's Manuscript. His solution for accents is usually taken as one of the best in Czech typography:



Finally, everyone can be extremely pleased with the results of František Štorm whose Storm Type Foundry brought during the last decade many beautiful fonts for moderate price (with the rise of OpenType perhaps one can fortunately forget about his quite unusable afm files). I find his accents the best!

žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!

His variant of Times in the last line is quite original and together with another rather good variant of Times from Macron, which sells Adobe fonts:

žluťoučký kuň úpěl ďábelské ódy!

could serve as a good inspiration for the T_EX Gyre project whose Termes from the narrow point of view of ‘palatal hook’ (the next fourth line) seems to be less acceptable . . .

žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!

On the other hand Pagella is much more acceptable than the following two versions of Palatino —

the first is from the Windows fonts directory, and the other from the older Czech localization of Adobe Palatino:

žluťoučký kuň úpěl ďábelské ódy!
 žluťoučký kuň úpěl ďábelské ódy!

Many other well-localised fonts of Adobe can be seen at [6].

Inspired by the ambitious and promising T_EX Gyre Project I humbly started some basic attempts with the Utopia family, which also belongs to the group of freely available fonts of Ghostscript and T_EX installations. I have liked to use Utopia for typesetting for a very long time (having previously implemented accented letters via virtual fonts generated with J. Zlatuška's program accents). I decided to try the following strategy:

- use Petr Olšák's program `a2ac` which adds composite characters into an afm file (of course with quoteright instead of appropriate accent);
- use `ttidy`, a small and rather old program (its version from [7] works only under the old DOS) which is able to transform composites in a Type 1 program into real characters;
- edit the shape and placement of ‘caron’ with help of Fontographer by hand;
- convert the result with MT1 package.

Acknowledgement. Many thanks to Bogusław — not least for his idea that this year is an appropriate time to celebrate 600 years of introducing accents into our languages . . .

References

1. Mirjam Bohatcová & kol.: Česká kniha v proměnách staletí (Czech book on changes over the centuries, in Czech), Panorama, Praha, 1990.
2. Oldřich Hlavsa: Book of type and design (Typografická písma latinková, in Czech), SNTL, Praha, 1960.
3. Bogusław Jackowski: Latin Modern fonts at the eleventh hour, talk at CSTUG meeting, Brno, 26 November, 2005.
4. Písmo ve výtvarné výchově (Types in Lessons of Aesthetics, in Czech), textbook for pedagogic faculties, SPN, Praha, 1989.
5. Karel Stejskal, Petr Voit: Illuminated manuscripts of the Hussite age (in Czech, English summary), National Library of Prague, Grafít, Praha, 1990.
6. www.caron.cz/pisma/pisma.html
7. fonts/utilities/titools/titidy on CTAN

Creation of a PostScript Type 1 logo font with MetaType 1

Klaus Höppner

Haardtring 230 a

64295 Darmstadt

Germany

klaus dot hoeppner (at) gmx dot de

Abstract

MetaType 1 is a tool created by Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk for creating PostScript Type 1 fonts. It uses METAPOST, t1utils and some AWK scripts to start from a METAPOST source with some special macros, resulting in the AFM, TFM and PFB files needed to use the font as any other PostScript font.

MetaType 1 was used to create the Latin Modern fonts, derived from Computer Modern fonts but including many more accented characters and nowadays part of most \TeX distributions. Other new fonts such as Iwona and Kurier have also been created by the developers of MetaType 1.

I came into contact with METAPOST when I wanted to convert an existing logo font from METAFONT to PostScript Type 1. Unfortunately there doesn't yet exist a tutorial or cookbook for using MetaType 1. So I started to play with the example fonts supplied as part of MetaType 1 and to read the comments in the source. This tutorial will give an example and the lessons I learned.

1 Introduction

When Donald E. Knuth invented \TeX , he also created his own description language for high quality fonts. It was named METAFONT. So the process from a \TeX source to some paperwork was as follows: Compile the \TeX source to get a DVI file that contains references to the fonts that were used in the document—in fact the only thing that \TeX knows about a font is its metrics. To produce the document on paper, the DVI driver invoked METAFONT (the program) to convert the METAFONT source of the font, i. e. the geometrical description of the font outlines, to a bitmapped font suited for the resolution and technical details of the printer by using the METAFONT mode for this special printer.

While this approach works fine if you work alone and just send your documents to your personal printer, it has some disadvantages if you want to exchange documents electronically. Normally, distributing DVI isn't the best idea, since it requires that the recipient has a \TeX system installed including all fonts that were used in your document—not to mention any graphics included in your document. So in most cases you will send a PostScript file or nowadays a PDF file. In this case, all the fonts from METAFONT sources will be embedded as bitmapped PostScript Type 3 fonts. When the recipient prints your document, it may look fine, but it may look poor if the METAFONT mode used to create the bit-

mapped font didn't match the printer, and the document will probably look very poor on the screen (especially in old versions of Acrobat Reader).

So when exchanging documents, it is preferable to embed the fonts as outline fonts. For these, the usual format used in the \TeX world is PostScript Type 1 (though this is gradually being replaced by OpenType). The Type 1 format uses a subset of the well established PostScript language.¹

Meanwhile, most of the fonts used in the \TeX world are available as PostScript Type 1 fonts, starting with the Type 1 version of Knuth's CM fonts up to the Latin Modern fonts that augment CM with a complete set of diacritic characters.

2 MetaType 1

MetaType 1 is the tool that was used to create the Latin Modern fonts from the METAFONT sources of CM fonts, and for the creation of completely new fonts such as Iwona.

MetaType 1 relies on METAPOST, a variant of METAFONT producing small pieces of PostScript as output, written by John Hobby. Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk wrote a set of METAPOST macros and added some AWK scripts to create the input files that can be con-

¹ It is sometimes said that Type 1 fonts are outline fonts while Type 3 are bitmap fonts. That's not true, since Type 3 fonts may comprise both outlines and bitmaps.

verted to Type 1 with `tlutils`. Thus, one advantage of MetaType 1 is that it uses a source format that is very similar to the old METAFONT sources.

3 Our example

I came into touch with MetaType 1 when I wasn't satisfied with the DANTE logo being typeset from the old METAFONT source with all the disadvantages mentioned above. So I wanted to give MetaType 1 a try to convert the DANTE logo font into a PostScript Type 1 font.

Fortunately, the DANTE logo font contains just the characters needed to set the logo:

dante

So, it was just five characters for which the METAFONT source had to be made suitable to be processed with MetaType 1.

Unfortunately, I found out that the available documentation for MetaType 1 was rather limited: articles from conference talks [1, 2], the commented source for the MetaType 1 macros and two sample fonts that are part of the MetaType 1 distribution.

But in the end, I found my way, and as you will see, was able to create my own Type 1 font. To make things a bit simpler for this tutorial, I will show the steps I made for a small test font with just two characters, “a” and “t”, simplified compared to the original characters from the DANTE logo font. Hopefully it will make the presented source more understandable, even if you haven't programmed in METAPOST before.

3.1 Installation

Installing MetaType 1 was easy enough. I downloaded the ZIP archive file from CTAN [3] and copied the files to the appropriate locations of my local `texmf` tree: the `.mp` files into `metapost/mt1`, the `.mft` files into `mft/mt1`, the `.sty` files into `macros/generic/mt1`, and finally the `.awk` and `.dat` files into `scripts/mt1`.²

The main problem in my case was that MetaType 1 was shipped with a set of DOS batch files that are used to create the fonts, but I was using GNU/Linux. So I looked into these files to find out what they do—in fact they were rather simple, just calling METAPOST to produce a small PostScript file for every glyph in the font and then using some AWK scripts to merge and assemble these files into a raw PostScript font that is converted into Post-

² This location isn't required since these files aren't found by the Kpathsea library, but instead via an environment variable, but at least this location seemed to be meaningful.

Listing 1: First definition of “a” and “t”.

```

encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
    scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
    yscaled 2radius shifted z1;
Fill pa;
unFill pb;
Fill pc;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
    -- (halfcircle rotated 180
        scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180
        scaled strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength
    yscaled strength shifted z3;
Fill pa;
Fill pb;
fix_hsbw(2hdist+5strength,0,0);
endglyph;

```

Script Type 1 with `tlasm` (part of `tlutils`). So several immediate files and steps are involved, but the workflow is straightforward. Eventually, I wrote a small Makefile that does the job on a Unix system, as shown in listing 3. From this point, I could create the TFM, PFB and MAP files for a font with the command `make FONT=myfont`.

I also manually created an FD file for using the font in L^AT_EX. These files could all be installed into the appropriate locations inside a `texmf` tree. Testing of a font is convenient in pdfT_EX since one can use a MAP file locally in a document using the `\pdfmapfile` primitive, while for a real font one normally will install the MAP file using the `updmap` script (or equivalent).

3.2 The first font

After these prerequisites were done, I could start with my first font. I copied the file `tapes.mp` (a sample font that is part of the MetaType 1 distribution) into `myfont.mp`, found several settings with font parameters starting with `pf_info_*`, changed them where appropriate (font name, family, creator, etc.) and kept the rest unchanged.

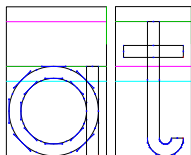
Then I defined the first two characters according to the following rule:

Characters consist of closed paths, filled or unfilled paths, where filled paths always turn counter clockwise and unfilled paths always clockwise.

So when designing the letter “a”, I defined an outer circle that was filled and then an inner circle to be unfilled and then a rectangular shape as vertical stem. And the letter “t” was just built from a vertical stem (with a hook at the right bottom) and a horizontal bar. The definitions for the characters are shown in listing 1.

Please notice in the definition of letter “a”, that the path for the outer circle is a (counter clockwise) fullcircle, while the inner circle is a reverse fullcircle, since the former one is filled while the latter one is unfilled. Filling and unfilling of the paths is done by the macros `Fill` and `unFill`; these macros warn you if the turning direction of the path is wrong.

Proofs for the glyphs are produced by compiling the file `myfont.mp` with `METAPOST`. As you can see, they really do look like an “a” and a “t”:



Now let’s see how the Type 1 font looks:



Something went wrong. After taking a closer look, it becomes obvious. The regions where filled paths overlap become unfilled. This is due to the fact that filling of paths is done with an *exclusive-or* fill, i.e. when filling a path, regions inside that are already black become white. As this isn’t what we want to achieve, we formulate another rule to keep in mind:

Paths must not overlap!

Although it is possible with pure `METAPOST` to find the intersection points of paths to remove overlapping parts, this tends to be painful. Since MetaType 1 was used to attach cedilla and ogonek accents to various characters in the extension of CM to LM, this painful work of finding the outline of two overlapping paths was encapsulated into a macro that is part of MetaType 1, named `find_outlines`. Let’s see how this macro is utilized for the letter “a”:

```
find_outlines(pa,pc)(r);
Fill r1;
```

It finds the outline of the two overlapping paths `pa` and `pc`, with the result written in the path array `r`. The result is an array because the outline of the paths may consist of more than one path, but in our case it is just one path, accessible as `r1`. The same is applied for the letter “t” (just the names of the two paths slightly differ).

When filling the new outlines instead of the overlapping paths, we now get the following result:



So, obviously finding the outline path for the “t” worked, but it failed for the “a”. Why? Because in the case of the “a”, both paths touch in one point without crossing at the right side of the vertical stem, i.e. they have an intersection point with the same direction vector. This confuses the macro that finds the outlines since it doesn’t know which path to follow — and in this case it chooses wrong. So, let’s bear in mind another rule:

Paths must not touch tangentially!

To resolve the problem, we use a simple trick: Shift the vertical stem a tiny amount to the right, so that the paths don’t touch anymore. In `METAPOST` you can use `eps` as a tiny positive number (in mathematics, an arbitrary small number is usually denoted by ϵ). The following lovely characters are the result (the `METAPOST` definitions are shown in listing 2):



3.3 Kerning

Our glyphs are ready, but a normal font has more features, such as kerning pairs and ligatures. In the

Listing 2: Definition of “a” and “t” with outlines.

```

encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginlyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
    scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
    yscaled 2radius shifted z1;
find_outlines(pa,pc) (r);
Fill r1;
unFill pb;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginlyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
    -- (halfcircle rotated 180
        scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180 scaled
        strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength
    yscaled strength shifted z3;
find_outlines(pa,pb) (r);
Fill r1;
fix_hsbw(2hdist+5strength,0,0);
endglyph;

```

former case, for a pair of characters the horizontal spacing between them is changed, while in the latter case a character pair is replaced by another glyph.

Defining a kerning pair in MetaType1 is simple. *After* the definition of the glyphs, we can add a kerning table. In our case it looks like this:

```
LK("a") KP("t")(-3ku); KL;
```

In the list of ligatures and kernings for the letter “a” we define a kerning of $-3ku$ if it is followed by the letter “t” to remove the optical gap between them (the kerning unit ‘ku’ is defined elsewhere in the METAPOST source). The effect of kerning is shown in figure 1.

Ligatures don’t make sense for our sample font,



Figure 1: Our font without (top) and with (bottom) kerning.

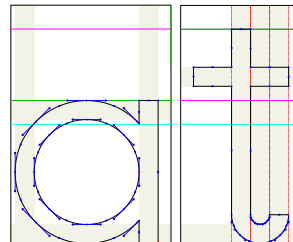


Figure 2: Hinting information (shaded areas).

so I leave them out for this tutorial. In principle they work similarly; you merely define from which slot in the font the replacement for a specified character pair is to be taken.

3.4 Hinting

When you embed fonts as outline fonts, you leave the task of rasterizing the glyphs to your output device (printer or viewer). Unfortunately, this final result may look rather poor, especially on low resolution devices such as screens. Imagine the letter “H” and how it is rasterized into pixels. If we’re unlucky, the left and right vertical stem will have a different width. On a printer with 1200 dpi it’s nearly unnoticeable, but on the screen a difference of one pixel makes it look quite ugly.

To prevent this, high quality fonts use a mechanism called “hinting” to help the rasterizer (e.g. the PostScript RIP in a printer) to keep vertical or horizontal stems the same width.

MetaType1 supports hinting by providing the macros `fix_hstem` and `fix_vstem` that try to find horizontal or vertical stems of a given width and add hinting information for them. For example, since we know that our letters “a” and “t” have stems of the width `strength`, we add hinting information by

```
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
```

You can see what hinting information was found as shaded areas in the proofs (figure 2).

4 Conclusions

I found that MetaType1 is a suitable tool to create PostScript Type 1 fonts. Though there is a lack of

beginning documentation, I was able to create a first font quite quickly by relying on an existing META-FONT source. Of course, knowledge of METAPOST or METAFONT is highly desirable. Understanding hinting is a bit more difficult, but finally possible.

References

- [1] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *MetaType 1: A MetaPost-based engine for generating Type 1 fonts*, Proc. of EuroTeX 2001, published in MAPS 26, 2001, 111–119. http://www.ntg.nl/maps/pdf/26_15.pdf
- [2] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Programming PostScript Type 1 fonts using MetaType 1: Auditing, enhancing, creating*, TUGboat, volume 24 (2003), no. 3. <http://tug.org/TUGboat/Articles/tb24-3/jackowski.pdf>
- [3] <http://ctan.org/fonts/utilities/metatype1/>

Listing 3: Makefile for font creation with MetaType 1.

```
METATYPE1 = /home/klaus/texmf/scripts/mt1

.PHONY: tfm pfb proof all

all: pfb tfm
proof: $(FONT).pdf
pfb: $(FONT).pfb
tfm: $(FONT).tfm

%.p: %.mp
    mpost "\generating:=0; \input $<"
    gawk -f $(METATYPE1)/mp2pf.awk \
        -vCD=$(METATYPE1)/pfcommon.dat \
        -vNAME='basename $< .mp'

%.pn: %.p
    gawk -f $(METATYPE1)/packsubr.awk \
        -vVERBOSE=1 -vLEV=5 -vOUP=$@ $<

%.pfb: %.pn
    t1asm -b $< $@

%.tfm: %.mp
    mpost "\generating:=1; \input $<"

%.pdf: %.ps
    ps2pdf $< $@

%.ps: %.dvi
    dvips -o $@ $<

%.dvi: %.tex
    tex $<

%.tex: %.mp
    mpost $<
    cp $< _t_m_p.mp
    mft _t_m_p.mp -style=mt1form.mft
    echo '\input mt1form.sty' > $@
    test -f piclist.tex && cat piclist.tex >> $@
    test -f _t_m_p.tex && cat _t_m_p.tex >> $@
    echo '\endproof' >> $@
```

Listing 4: The complete font.

```
% A sample font
input fontbase;

% Global parameters for all characters
size := 1000; depth := 0; math_axis := 1/2size;
radius := 300; height := 900; strength := 80;
ku := 18; hdist := 3ku; round_hdist := 1ku;

% Font settings
pf_info_familyname "MyFont";
pf_info_fontname "MyFont-Regular";
pf_info_weight "Normal";
pf_info_version "0.01";
pf_info_capheight height;
pf_info_xheight 2radius;
pf_info_space 10ku;
pf_info_adl size, 0, 0;
pf_info_author "Made by KH"
pf_info_overshoots (1000,10), (0, -10);
pf_info_encoding "at";
pf_info_creationdate;

beginfont

encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle scaled (2radius-2strength)
    shifted z0;
pc = unitsquare xscaled strength yscaled 2radius
    shifted z1;
find_outlines(pa,pc)(r);
Fill r1; unFill pb;
fix_hstem(strength,pa,pb,pc);
fix_vstem(strength,pa,pb,pc);
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1 -- (halfcircle rotated 180
    scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180
    scaled strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength yscaled strength
    shifted z3;
find_outlines(pa,pb)(r);
Fill r1;
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
fix_hsbw(2hdist+5strength,0,0);
endglyph;

LK("a") KP("t")(-3ku); KL;
endfont.
```


Procedures for font comparison

Karel Píška

Institute of Physics of the ASCR, v.v.i.

CZ-182 21 Prague, Czech Republic

piska (at) fzu dot cz

Abstract

This contribution presents several programs: Linux standalone scripts for comparison and visualization of font elements. Simplified versions of the programs are restricted to use with the fonts already installed in a \TeX system and show, for example, how to solve the following tasks:

- Comparison of two bitmapped or outline representations of a glyph pair at two different resolutions or from two related \TeX fonts using a color mix technique in \pdfTeX .
- Comparison of kerning pairs in two (or three) related \TeX fonts or in two releases of one font.
- Comparison of a glyph pair in two versions or in two outline fonts, analyzing sequences of cubic curve segments in font programs.

The programs will be demonstrated by several examples.

1 Common characteristics

The purpose of the `tfcpr` package, containing procedures for \TeX font comparison, is to provide tools for checking and comparison of the fonts used in a \TeX environment. This paper is a complementary work to the article “Font verification and comparison in examples” [9]. The programs are written in the form of Linux scripts, written in the `bash` shell language. After download they are unzipped into a local working directory. The simplest method is to start the programs locally with the `./` prefix. They invoke various programs from Linux or its \TeX subsystem, such as `gawk` [6], `METAFONT`, `LA \TeX` or `pdfLA \TeX` , `dvips`, etc., and Acrobat Reader is usually called in the final step. They read the font files: metrics (`tfm`), `METAFONT` sources, Type 1 (`pfb`) or OpenType (`otf`); generate intermediate font bitmaps (`pk`) and PostScript; and produce PDF documents. The `kpathsea` library is used to find font files. Uninstalled fonts, including unused versions or releases, should be copied for testing purposes also into the current local directory. Additionally, we have to change their file names to avoid name collisions if it is necessary to distinguish them from fonts already existing in our system. `FontForge` [8] and `t1utils` [7] are also used for preprocessing of data from outline fonts.

The real testing software is under permanent revision depending on a font family being processed at the moment, since it must often be modified for

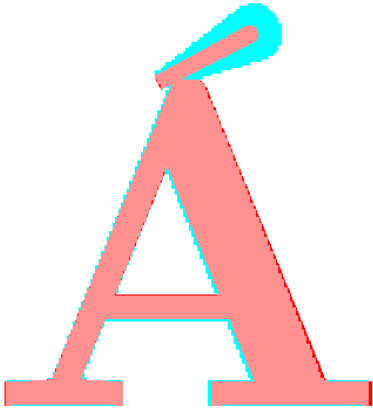
various encoding schemes and naming conventions; usually all glyphs and all character pairs for a given encoding should be processed. It has no general or common user interface and it would be too difficult and expensive to transform it to a transparent and computer independent form. The programs presented here are simplified excerpts restricted to processing of a single glyph or a character pair and extended with a user interface to call them via commands and produce the visual output in PDF. They can be used quickly. On the other hand, they do not work efficiently. In most examples, fonts from the Computer Modern [2, 3], CS fonts [4], and the Latin Modern [5] families are presented.

2 Glyph comparison for bitmapped fonts

The aim in comparing a `METAFONT` font and its Type 1 version at various resolutions is to check the correctness of the font design in `METAFONT` and its proper conversion into an outline font format.

We can compare glyph images for one `METAFONT` font at two different resolutions or the bitmapped representation with the Type 1 version of the font converted from its `METAFONT` sources.

The execution of the command
`./cprpk cmr10 97 1200 2400`
starts by generating bitmaps with `METAFONT`. The four parameters define the font name, the decimal (or octal, for example `\'141`) character code and

Figure 1: `./cprpk csbx10 193 1200 5333`Figure 3: `./cprpkt1 cmr10 97 1200`Figure 2: `./cprpk cmr10 97 1200 2400`Figure 4: `./cprpkt1c cmr10 97 1200`

two resolutions. Then we produce two glyph instances for both resolutions distinguished by postfix “a” or “b” with pdf \LaTeX —see the example document in Fig. 6. After that, again with pdf \LaTeX we mix both components into one picture (Fig. 7). All these \LaTeX sources have been generated automatically and dynamically according to the command. The final results are shown in Figure 1: the bit-mapped `csbx10` “A” at 1200 dpi and 5333 dpi; and in Fig. 2: the Computer Modern Roman `cmr10` “a” at 1200 dpi and 2400 dpi. The first resolution glyph image is filled with cyan (light), the second one or Type 1 is filled (stroked) with red (dark), and the common area is in pink (looks lightest). (For the printed issue, the figures have been manually converted to grayscale.) The Type 1 counterpart can substitute one bitmap (Fig. 3) or may be expressed in the contour mode “1 Tr”—see figures 4 and 5. The figure captions contain the commands generating the corresponding pictures.

Figure 5: `./cprt1cpk cmmib5 55 2400`

```
\documentclass{article}\pagestyle{empty}\usepackage{graphicx}
\font\fa=cmr10a
\begin{document}\scalebox{40}{\fa\char97}\end{document}
```

```
\documentclass{article}\pagestyle{empty}\usepackage{graphicx}
\font\fb=cmr10b
\begin{document}\scalebox{40}{\fb\char97}\end{document}
```

Figure 6: Generating of glyph images for comparison.

```
\documentclass{article}
\usepackage{graphicx}
\def\Default{\pdfliteral{0 g 0 G}}
\pdfpageresources
{/ExtGState
  << /Luminosity << /Type /ExtGState /BM /Luminosity >> >>}
\def\Acolor{0 1 1 rg 0 1 1 RG}% cyan
\def\Bcolor{1 0 0 rg}% red
\begin{document}
\setlength{\fboxsep}{0pt}\setlength{\fboxrule}{0pt}
\begin{figure}\begin{center}
\makebox[0pt][l]{\pdfliteral{/Luminosity gs \Bcolor}%
\includegraphics[viewport=90 310 470 710]{cmr10_97-2400.pdf}}%
{\pdfliteral{\Acolor}%
\includegraphics[viewport=90 310 470 710]{cmr10_97-1200.pdf}}
\Default\label{cmr10_97}
\caption{cmr10: 97 {\pdfliteral{\Acolor} 1200pk}
{\pdfliteral{\Bcolor} 2400pk}}
\Default
\end{center}\end{figure}
\end{document}
```

Figure 7: A color mix of two glyph instances.

```
\documentclass{article}
\newlength{\bbox}\newlength{\cbox}%
\def\fboxsep{0pt}\def\fboxrule{0.1pt}
\usepackage{ifthen}
\def\krule#1{%
\ifthenelse{\lengthtest{#1>0pt}}{\rule{10#1}{1ex}{$>$}}{%
\ifthenelse{\lengthtest{#1<0pt}}{${<}$}{\rule{-10#1}{1ex}}{}}
\def\paira#1#2#3{%
\font\fna=#1
\fna\settowidth{\bbox}{#2#3}%
\settowidth{\cbox}{\mbox{#2}\mbox{#3}}%
\addtolength{\bbox}{-\cbox}%
\fbox{#2}\kern-0.2pt\kern\bbox\fbox{#3}\fbox{#2#3}
#1 \krule{\bbox}
\\}
\begin{document}
\noindent
\paira{cmr10}{k}{a}
\paira{csr10}{k}{a}
\paira{ec-lmr10}{k}{a}
\end{document}
```

Figure 8: Generating a test for three kerning pairs.

```

./prfkrna k a cmr10 csr10 ec-lmr10
./prfkrna K O cmr10 csr10 ec-lmr10
./prfkrna v a cmr10 csr10 ec-lmr10
./prfkrna A c cmr10 csr10 ec-lmr10
./prfkrna P , cmr10 csr10 ec-lmr10
./prfkrna T . cmr10 csr10 ec-lmr10
./prfkrna f ! cmr10 csr10 ec-lmr10
./cprkrna A C cmti10 ec-lmri10
./cprkrna i i cmcsc10 ec-lmcsc10
./cprkrna I I cmcsc10 ec-lmcsc10

```

Figure 9: Sample commands for kerning tests.

3 Comparison of kerning pairs

Our next aim is to verify compatibility between two or more related T_EX fonts. Kerning data are taken from the corresponding metric files (`tfm`). The first example (Fig. 8), invoked by the command `./prfkrna k a cmr10 csr10 ec-lmr10` compares kerning of character pairs in three related fonts, extracting the first lines of the final output (Fig. 10). Figure 9 shows a sample command list.

The L^AT_EX source in Fig. 11, generated by the command line

```

./cprkrna i i cmcsc10 ec-lmcsc10,

```

is connected with the last lines in figures 9 and 10 and, unlike the previous examples, we really do compare the kern values between two instances of the character pairs. The equal sign “=” in the CMCS10 line (second from the bottom in Fig. 10) signals the identical kerns, while the rule in the EC-LMCS10 line (bottom line) denotes the common kern value in both fonts. The varying widths of the rules relates to the different kerns and their relative ratio.

The table in Fig. 12 summarizes several selected differences in the kerning pairs for three font families expressed in the kern degree units (e.g., “k#”).

For example, for Latin Modern in the T₁/EC encoded metrics we have:

```

kerning pairs absent in LM: "k": "a" and "K": "0";
LM compatible with CM: "v": "a" and
    "p", "T": ". , , ";
a new kerning pair in LM: "A": "c";
a kern doubled in LM: "A": "C" in italics;
other changes in LM: "i": "i" in small caps or
    "f": "!",

```

k**a** cmr10 <■
k**a** csr10 <■
k**a** ec-lmr10
K**O** cmr10 <■
K**O** csr10 <■
K**O** ec-lmr10
v**a** cmr10 <■
v**a** csr10 <■
v**a** ec-lmr10 <■
A**c** cmr10
A**c** csr10
A**c** ec-lmr10 <■
P**,** cmr10 <■
P**,** csr10 <■
P**,** ec-lmr10 <■
T**.** cmr10
T**.** csr10 <■
T**.** ec-lmr10
f**!** cmr10 ■>
f**!** csr10 ■>
f**!** ec-lmr10 ■>
A**C** *cmti10* <■
A**C** *ec-lmri10* <■
i**i** CMCS10 ■>
i**i** EC-LMCS10 ■>
I**I** CMCS10 =
I**I** EC-LMCS10 ■>

Figure 10: Several tests of kerning pairs.


```

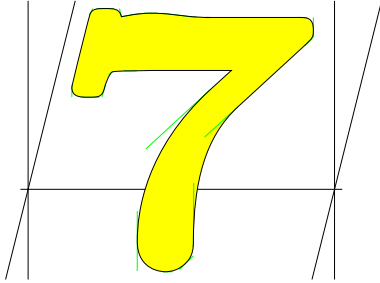
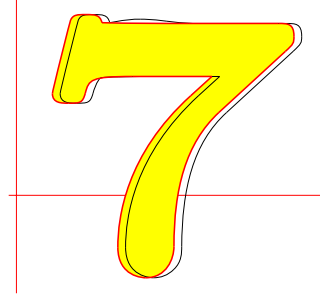
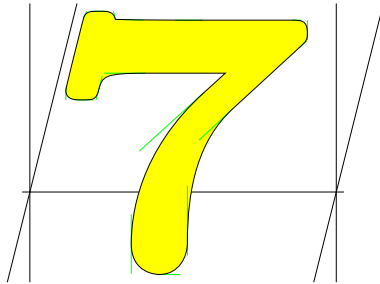
\documentclass{article}\font\fna=cmcsc10\font\fnb=ec-lmcsc10
\newlength{\bbox}\newlength{\cbox}%
\newlength{\bxa}\newlength{\bxb}%
\def\fbboxsep{0pt}\def\fbboxrule{0.1pt}
\usepackage{ifthen}
\def\krule#1{%
\ifthenelse{\lengthtest{#1>0pt}}{\rule{10#1}{1ex}{$=$}}{}%
\ifthenelse{\lengthtest{#1<0pt}}{\{$<$}\rule{-10#1}{1ex}}{}%
\def\pair#1#2#3#4{%
\fna\settowidth{\bxa}{#1#2}%
\settowidth{\cbox}{\mbox{#1}\mbox{#2}}%
\addtolength{\bxa}{-\cbox}%
\fnb\settowidth{\bxb}{#3#4}%
\settowidth{\cbox}{\mbox{#3}\mbox{#4}}%
\addtolength{\bxb}{-\cbox}%
\ifthenelse{\lengthtest{\bxa = \bxb}}%
{\fna\fbbox{#1}\kern-0.2pt\kern\bxa\fbbox{#2}\fbbox{#1#2}
cmcsc10 {$=$}
}\
{\fnb\fbbox{#3}\kern-0.2pt\kern\bxb\fbbox{#4}\fbbox{#3#4}
ec-lmcsc10 \krule{\bxb}
\}}
{\fna\fbbox{#1}\kern-0.2pt\kern\bxa\fbbox{#2}\fbbox{#1#2}
cmcsc10 \krule{\bxa}
}\
{\fnb\fbbox{#3}\kern-0.2pt\kern\bxb\fbbox{#4}\fbbox{#3#4}
ec-lmcsc10 \krule{\bxb}
\}}
\begin{document}
\noindent
\pair{i}{i}{i}{i}
\end{document}

```

Figure 11: Comparison of two instances of a kerning pair.

	cmr10	csr10	ec-lmr10	
"k": "a" kern	2k=-u	k	0	CM<CS<EC=0
"K": "0" kern	k	k	0	CM<CS<EC=0
"v": "a" kern	2k=-u	k	2k=-u	EC=CM<CS=k
"A": "c" kern	0	0	k	EC<CM=CS=0
"P": "." kern	3k=kk	k	3k=kk	EC=CM<CS=k
"P": ", " kern	3k=kk	k	3k=kk	EC=CM<CS=k
"T": "." kern	0	k	0	CS<CM=EC=0
"T": ", " kern	0	k	0	CS<CM=EC=0
"f": "!" kern	-2.8k	-2.8k	-k	0<EC<CM=CS
	cmti10	csti10	ec-lmri10	
"A": "C" kern	k	k	2k=-u	EC<CM=CS=k
	cmcsc10	cscsc10	ec-lmcsc10	
"i": "i" kern	-0.76k	-0.76k	-3.05k	0<CS=CM<EC

Figure 12: Examples of differences between kerns.

Figure 13: `./prfof cmmib5 sevenoldstyle`Figure 15: `./cprofof cmmib5 lmmib5 sevenoldstyle seven.taboldstyle`Figure 14: `./prfof lmmib5 seven.taboldstyle`

4 Proofs and comparison of outline fonts

The next tools are `prfof`, which produces proof-sheets for single glyphs, and `cprofof`, which compares glyphs from Type 1 or OpenType fonts. The font editor FontForge [8] is used to parse fonts, that is, for unpacking, extraction and export of a partial font and glyph data. Type 1 fonts are also processed with the `t1utils` package [7].

The examples in figures 13 and 14 demonstrate the “old style seven” in Computer Modern `cmmib5` and in Latin Modern `lmmib5` and its comparison in both representations (Fig. 15).

We must note that the Type 1 version converted by Blue Sky and Y&Y is older than the latest modifications of the CM sources. Please compare also with Fig. 5 where the METAFONT definitions have been corrected while the Type 1 outline curves are still wrong.

4.1 Comparison of different releases of the same font

An older font version can be unzipped and copied into the current directory; its file names must be renamed before starting comparison with the actual production font release in the installed TDS tree. Here is a sample approach:

```
LM1="lm1.00bas.zip"
PFB="fonts/type1/public/lm"
unzip -j $LM1 $PFB/lmri10.pfb
mv lmri10.pfb lmri10a.pfb
./prfof lmri10a perthousand
./prfof lmri10 perthousand
./cprofof lmri10a lmri10 perthousand
```

4.2 Change list

A helpful idea is to assemble a change list: a list of the font and glyph names with known bugs or changes, and run it with a previous version and then with the current font release to confirm the changes have been done and the bugs have been fixed. An example of such a list is presented in Fig. 16.

```
lmri10 perthousand
lmri10 permyriad
lmu10 perthousand
lmu10 permyriad
lmu10 degree
lmdunh10 uring
lmduno10 uring
lmmib5 seven.taboldstyle
lmmib7 seven.taboldstyle
lmmi5 seven.taboldstyle
lmtcsc10 F
lmtcsc10 I
lmtcsc10 Iacute
lmtcso10 F
lmtcso10 I
lmtcso10 Iacute
```

Figure 16: A change list for LM.

5 Availability

The `tfcpr` package can be downloaded from <http://www-hep.fzu.cz/~piska/tfcpr.html>. The programs can be distributed as “public domain software”, may be freely used (without warranty), corrected, modified, adapted or included in other packages.

References

- [1] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986. Volume C of *Computers and Typesetting*.
- [2] Donald E. Knuth. *Computer Modern Typefaces*. Addison-Wesley, 1986. Volume E of *Computers and Typesetting*.
- [3] Computer Modern fonts. CTAN:/fonts/cm.
- [4] CS fonts. <ftp://math.feld.cvut.cz/pub/cstex/base/csfonts.tar.gz>.
- [5] Latin Modern fonts. CTAN:/fonts/lm.
- [6] Free Software Foundation. GNU awk, <http://www.gnu.org/software/gawk>.
- [7] Eddie Kohler. `tlutils` (Type 1 tools), <http://freshmeat.net/projects/tlutils>.
- [8] George Williams. Font creation with FontForge. *EuroTEX 2003 Proceedings, TUGboat*, 24(3):531–544, 2003; <http://fontforge.sourceforge.net>.
- [9] Karel Píška. Font verification and comparison in examples, *EuroTEX 2006 Proceedings, TUGboat* 27(1):71–75, 2006.

A Appendix: tfcpr: Synopsis and examples

A.1 Glyph comparison for bitmapped fonts

```
cprpk font code res1 res2
cprpk cmr10 97 1200 2400
cprpk cmr10 \'141 1200 2400
```

```
cprpkt1 font code res
cprpkt1 cmr10 97 1200
cprpkt1 cmr10 \'141 1200
```

```
cprpkt1c font code res
cprpkt1c cmr10 97 1200
cprpkt1c cmr10 \'141 1200
```

```
cprt1cpk font code res
cprt1cpk cmr10 97 1200
cprt1cpk cmr10 \'141 1200
```

```
cprpkpk font1 font2 code1 code2 res1 res2
```

font a T_EX font name, common to .mf, .tfm, pk
code may be decimal or octal (starts with \')
 Predefined resolutions: 300 600 1200 2400 2602 5333

A.2 Comparison of kerning pairs

```
prfkrn font code1 code2 [font code3 code4]...
prfkrn cmr10 65 99 ec-lmr10 65 99
prfkrn cmr10 \'101 \'143 ec-lmr10 \'101 \'143
```

```
prfkrna char1 char2 font [font]...
prfkrna A c cmr10 ec-lmr10
```

```
cprkrn font1 code1a code1b font2 code2a code2b
cprkrn cmr10 65 97 ec-lmr10 65 97
cprkrn cmr10 \'101 \'143 ec-lmr10 \'101 \'143
```

```
cprkrna char_a char_b font1 font2
cprkrna A c cmr10 ec-lmr10
```

font a T_EX font name, i.e., .tfm
char an ASCII symbol recognized by T_EX

A.3 Proofing and comparison for outline fonts

```
prfof font glyphname
prfof cmmib5 sevenoldstyle
cprof font1 font2 glyphname1 [glyphname2]
cprof cmmib5 lmmib5 sevenoldstyle \
      seven.taboldstyle
```

font is *name*[.pfb] or *name*.otf, i.e.,
 only .pfb may be omitted
glyphname is PostScript or OTF, according to context

Comments and suggestions about the Latin Modern fonts

Karel Píška

Institute of Physics of the ASCR, v.v.i.

CZ-182 21 Prague, Czech Republic

piska (at) fzu dot cz

Abstract

This contribution:

- describes a process of verification of the Latin Modern fonts and lists selected aspects (typographic and technical) tested during this activity;
- summarizes the results of checking and comparisons, mostly for version 0.99.3 (2005) and 1.00 (2006), while information about the current version 1.010/x (2007) is limited;
- documents bugs and their correction, often in a visual form;
- remarks on the crucial changes in the recent versions (2006 and 2007);
- compares individual glyph shapes and finds differences in the last Type 1 and OpenType releases;
- compares metrics, especially character widths and kerning pairs, between the T1 (EC) and CS (CM) encoded subsets, and between ver. 0.99.3 and ver. 1.00, analyzing compatibility and listing the differences;
- studies accents and accented letters, i.e. accent shapes and their positioning, mainly for the accented characters common to CS (Czech and Slovak font collection) and T1 (EC);
- discusses problems with accurate and optimal outline representations, as the Latin Modern font family is a descendant of Computer Modern (designed in METAFONT) and converted into an outline approximation with cubic curves;
- comments on hinting strategies, particularly for accents and slanted fonts.

1 Introduction

The purpose of the activities described in this article was to improve the Latin Modern font package, helping to change unintentional features and fixing mistakes, with a special emphasis on preserving compatibility in typesetting Czech and Slovak documents with L^AT_EX using the Latin Modern outline fonts to substitute for older extensions of the Computer Modern typefaces.

The current version of the paper is a partial excerpt of the technical documentation, a collection of visual documents, mostly in PDF. All are still undergoing revision, trying to reflect the actual updates of the LM package, which is possible only with some time delay. I think it is not unreasonable to include tests of the previous release of LM, performed and stored in “my” archive last year.

I collected my comments, reports and proposals on my web site:

<http://www-hep.fzu.cz/~piska/lm2005.html>

<http://www-hep.fzu.cz/~piska/lm2006.html>

I have specially added some longer tables (OTF glyph names, kerning pairs) to be available in a printed form for investigation and discussion.

For user information about LM, we recommend “An exploration of the Latin Modern fonts”, an article written by Will Robertson [13].

2 Global remarks about font verification

The multistep checking process of an upcoming release can be divided into several stages.

1. We start the first stage from a survey to determine significant changes in comparison with the previous version or releases, important differences, extensions or exclusions. And then we generate “primary” proof printings, for example, the complete proof sheet pages for all fonts and all glyphs present in the font family; all ligature and kerning pairs for all fonts and for selected encodings. After some adaptation of the programs involved, this output can usually be produced automatically with only minimal assistance.

ver.	date	comments about crucial changes
0.99.3	28 Oct 2005	
1.00	13 Apr 2006	metrics completely recalculated
1.010	16 Jan 2007	glyph names changed in OpenType
1.010a	23 Feb 2007	
1.010x	28 Feb 2007	

Table 1: Overview of recent releases of LM.

Quite a lot of disk space to store many huge files is needed. A rich archive of PDF documents is created and ready for human visual scanning with Acrobat Reader to search glyph images for evident errors, to study “usual” (from previously known opinions) weak points, zooming the typical parts of glyphs where artifacts may occur; to search for unsuitable kerning pairs or improper kerns; and to study consistency or compatibility of associated font elements.

The test printings covering the entire glyph repertoire and all kerning pairs for T1 (EC) encoding were generated in 2006 for LM 1.00.

- In the next stage we compare the actual font release with the preceding release(s), e.g. we find differences between both instances of all glyphs in their outline curve representation. We also compare the major metric data important for typesetting with \TeX , especially the character widths, kerning and ligature pairs.
- The subsequent stage depends on the results of the previous analysis. We select, study, select and study again chosen features, potential mistakes and strange events in detail to detect and localize bugs, and give a classification or conclusion, to prepare a report in textual and/or visual form.
- A comparison with related and other relevant fonts may be important to confirm identity and compatibility or to find differences, intentional or unintentional.
- Finally, we perform an overall evaluation of the available data, summarize the results, and compile a document with visual demonstration and written comments, conclusions, reports, suggestions and recommendations.

3 Developments and changes in Latin Modern

To begin with, Table 1 shows a concise summary of recent releases of LM.

Text fonts					
		ver. 1.00		ver. 1.010[x]	
	#n	#g0	#k0	#g1	#k1
b	10x	701	9399	742	9344
bi	1x	701	12134	742	11963
sc	2x	692	8742	735	8676
r	21x	701	9413	742	9358
ri	6x	701	12148	742	11977
ss	14x	701	8732	742	8677
sq	4x	704	8732	745	8677
tt	14x	662	0	703	0
tc	2x	659	0	702	0
Subtotal text fonts:					
	72	49914	551345	52874	547321
Mathematical fonts					
sy	9x	132	26	132	26
ex	1x	130	0	130	0
mi	10x	130	164	130	164
Subtotal math fonts:					
	20	2618	1874	2618	1874
Total LM fonts:					
	92	52532	553219	55492	549195
#n number of fonts with the same counts					
#g number of glyphs					
#k number of kerning pairs					

Table 2: Numbers of fonts and glyphs.

Table 2 presents the numbers of glyphs and kerning pairs in LM ver. 1.00 (#g0 and #k0) and the current ver. 1.010[x] (#g1 and #k1), subtotal counts for 72 text fonts and 20 mathematical fonts and the total sums, where:

b = (Roman) Bold, Demi; bi = (Roman) BoldItalic; sc = SmallCaps, r = (Roman) Regular, Oblique, TypewriterVarWd; ri = (Roman) Italic, Unslanted; ss = Sans; sq = SansQuotation; tc = TypewriterCaps; tt = Typewriter; sy = MathSymbols; ex = MathExtension; mi = MathItalic.

One very important modification in LM ver. 1.010 is the change of glyph names in OpenType. In Table 3, Unicode code points are listed together with the OpenType (Unicode) glyph names (in the second column) and PostScript (Type 1) names (column 3). This information has been added to give a quick explanation of the “unintelligible” OTF glyph names in a short and compressed form.

4 Comparison of releases

Because the PostScript and Type 1 glyph names are not identical starting with the version released in 2007 it is impossible to compare the glyphs by name

F6C9	acute.cap Acute	EA14	space_uni0309 hookabove
2116	afii61352 nomero	EA13	space_uni0309.cap Hookabove
2217	asterisk.math asteriskmath	EA17	space_uni030A_uni0301 ringacute
EFEE	breve.cap Breve	EA16	space_uni030A_uni0301.cap Ringacute
F6CA	caron.cap Caron	F6D3	space_uni030F dblgrave
EFF7	circumflex.cap Circumflex	F6D6	space_uni030F.cap dblGrave
-	copyright.var varcopyright	EA07	space_uni0311 breveinverted
F6CB	dieresis.cap Dieresis	EA06	space_uni0311.cap Breveinverted
EFED	dotaccent.cap Dotaccent	EB19	space_uni0323 dotbelow
-	dotaccent.var vardotaccent	EA08	space_uni032F breveinvertedlow
1E0C	D_uni0323 Ddotbelow	EB69	space_uni0330 tildelow
1E0D	d_uni0323 ddotbelow	EB09	star.alt born
FB00	f_f ff	EB2A	S_S Germandbls
FB03	f_f_i ffi	EFF5	tilde.cap Tilde
FB04	f_f_l ffl	1E6C	T_uni0323 Tdotbelow
FB01	f_i fi	1E6D	t_uni0323 tdotbelow
FB02	f_l fl	00A0	uni00A0 nbspace
F6CE	grave.cap Grave	00AD	uni00AD sfthyphen
F6CF	hungarumlaut.cap Hungarumlaut	0218	uni0218 Scommaaccent
1E24	H_uni0323 Hdotbelow	0219	uni0219 scommaaccent
1E25	h_uni0323 hdotbelow	021A	uni021A Tcommaaccent
-	I.var varI	021B	uni021B tcommaaccent
-	Iogonek.var varIogonek	0300	uni0300 gravecomb
0132	I_J IJ	E300	uni0300.cap Gravecomb
-	I_J.var varIJ	0301	uni0301 acutecomb
0133	i_j ij	E301	uni0301.cap Acutecomb
F6BE	j.dotless dotlessj	0302	uni0302 circumflexcomb
1E36	L_uni0323 Ldotbelow	E302	uni0302.cap Circumflexcomb
1E37	l_uni0323 ldotbelow	0303	uni0303 tildecomb
1E39	l_uni0323_uni0304 ldotbelowmacron	E303	uni0303.cap Tildecomb
1E38	L_uni0323_uni0304.cap Ldotbelowmacron	0304	uni0304 macroncomb
F6D0	macron.cap Macron	E304	uni0304.cap Macroncomb
1E42	M_uni0323 Mdotbelow	0306	uni0306 brevecomb
1E43	m_uni0323 mdotbelow	E306	uni0306.cap Brevecomb
1E45	n_uni0307 ndotaccent	0307	uni0307 dotaccentcomb
1E44	N_uni0307.cap Ndotaccent	E307	uni0307.cap Dotaccentcomb
1E46	N_uni0323 Ndotbelow	0308	uni0308 dieresiscomb
1E47	n_uni0323 ndotbelow	E308	uni0308.cap Dieresiscomb
-	registered.var varregistered	0309	uni0309 hookabovecomb
EFF3	ring.cap Ring	E309	uni0309.cap Hookabovecomb
1E59	r_uni0307 rdotaccent	030A	uni030A ringcomb
1E58	R_uni0307.cap Rdotaccent	E30A	uni030A.cap Ringcomb
1E5A	R_uni0323 Rdotbelow	030B	uni030B hungarumlautcomb
1E5B	r_uni0323 rdotbelow	E30B	uni030B.cap Hungarumlautcomb
1E5D	r_uni0323_uni0304 rdotbelowmacron	030C	uni030C caroncomb
1E5C	R_uni0323_uni0304.cap Rdotbelowmacron	E30C	uni030C.cap Caroncomb
2423	space.visible visiblespace	030F	uni030F dblgravecomb
EA0E	space_uni0302_uni0300 circumflexgrave	E30F	uni030F.cap Dblgravecomb
EA0D	space_uni0302_uni0300.cap Circumflexgrave	0311	uni0311 breveinvertedcomb
EA0C	space_uni0302_uni0301 circumflexacute	E311	uni0311.cap Breveinvertedcomb
EA0B	space_uni0302_uni0301.cap Circumflexacute	0323	uni0323 dotbelowcomb
EA12	space_uni0302_uni0303 circumflextilde	0326	uni0326 commaaccentcomb
EA11	space_uni0302_uni0303.cap Circumflextilde	032E	uni032E brevelowcomb
EA10	space_uni0302_uni0309 circumflexhookabove	032F	uni032F breveinvertedlowcomb
EA0F	space_uni0302_uni0309.cap Circumflexhookabove	F6DE	uni2014.alt1 threequartersemdash
EA03	space_uni0306_uni0300 brevegrave	EB6D	uni2014.alt2 twelvedash
EA02	space_uni0306_uni0300.cap Brevegrave	2127	uni2127 mho
EA01	space_uni0306_uni0301 breveacute	2190	uni2190 arrowleft
EA00	space_uni0306_uni0301.cap Breveacute	2191	uni2191 arrowup
EA0A	space_uni0306_uni0303 brevetilde	2192	uni2192 arrowright
EA09	space_uni0306_uni0303.cap Brevetilde	2193	uni2193 arrowdown
EA05	space_uni0306_uni0309 brevehookabove	266A	uni266A musicalnote
EA04	space_uni0306_uni0309.cap Brevehookabove		

Table 3: OTF and PostScript glyph names.

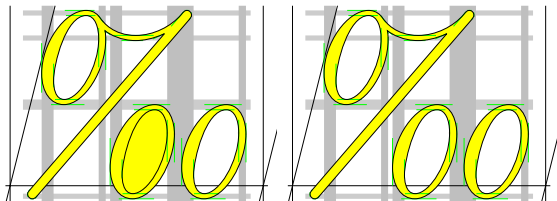


Figure 1: `lMRI10:perthousand` before and after correction of path directions.

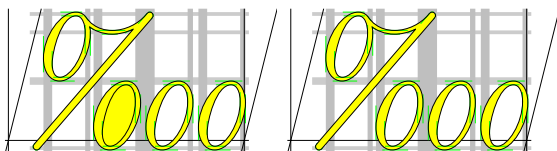


Figure 2: `lMRI10:permyriad` before and after correction of path directions.

without preparation of a new program to do a complete glyph comparison in a different way. The tests performed last year with LM 1.00 may be interesting, but they are out-of-date for this article.

5 Bug reports and corrections

It is important to demonstrate bugs present in the last release and then confirm they have been fixed in the current release. The technical documentation contains a comprehensive list of bugs and other problematic events. Here we will show several examples showing glyph corrections, improvements or other changes (figs. 1 and 2).

Single tests to examine glyphs in the actual fonts can be performed with procedures from my package `tfcp` [12].

6 Metrics: compatibility and/or quality

One major task, supported by CSTUG, was exploring use of LM for the characters in the common subset of the standard Computer Modern and CS fonts, mainly accented letters belonging to the Czech and Slovak character set.

For all characters from the intersection of the `ec-lm` encoding and CS (which covers all characters from CM) we compared metric data: character widths in the corresponding `tfm` metric files and also with their equivalents in `pfb`, `afm` and `otf`, and kerning pairs in `tfm`.

Generally, good agreement in the glyph widths was found, the differences being negligible (in the last digits), and due to internal numeric representation.

After the complete recalculation of metrics included in LM ver. 1.00 I have found only a couple

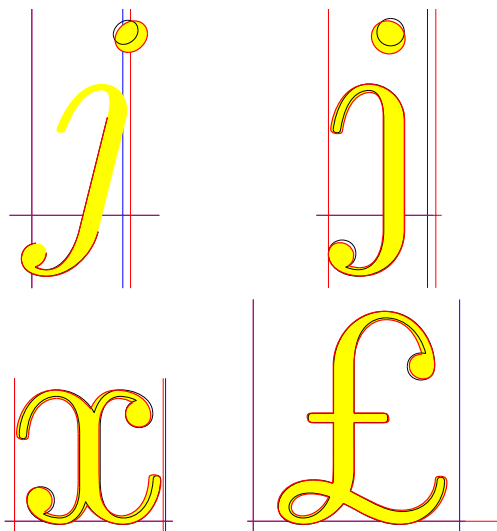


Figure 3: Different widths (and shapes) in `lMRI10:j` and `lmu10:j`, `x`, `sterling`.

of cases of changing character widths: “j” in `lMRI*`, and “x” and `sterling` in `lmu10` (fig. 3). And a question: Should have the dots in “i”, “j” and “ij” have identical shapes and sizes, or may they differ?

The kerning pair lists were (re)generated by the authors of LM with a semi-automatic algorithm reflecting and adjusting the horizontal spaces between the adjacent characters. All the character pairs from the given subset (defined for the corresponding encoding) were included in the processing. However, the numbers of the kerning pairs in the metric files seem to be extremely large and probably many of these kerning pairs are not relevant to any language. I have no good idea how to exclude *automatically* the irrelevant kerning pairs to reduce the space needed for metric data. I have decided to include the complete list of the ligature and kerning pairs for `lMRI10` (Latin Modern Roman at 10 pt); see Tables 4 and 5, in the hope that readers will respond with their comments and suggestions.

We pay special attention to the present or absent kerning pairs and to the kern values in the T1 (`ec-lm`) encoding in a comparison with CM and CS fonts. The kerning pairs can be reordered and divided into groups according their agreement or disagreement between CM, CS and LM (for the `ec-lm` encoding) and we can list cases of their discarding, additions or changes in the associated fonts from these families. The corresponding data are collected in my “technical documentation” which I am gathering step by step. Several selected examples are presented in my articles about font verification and comparison [11, 12].

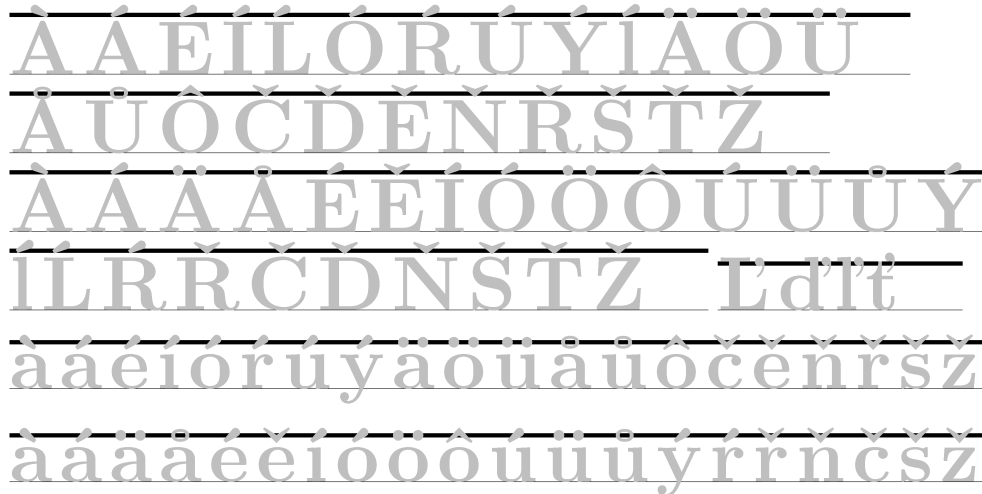


Figure 4: Common optical accent heights.

An important remark: *In OpenType all metric data must be integers* (e.g. metric dimensions). Therefore, metrics converted from TeX fonts have to be rounded. (Other outline fonts have been already designed on an integer coordinate grid and the TeX metrics preserve their values.) I cannot estimate the real effects of this incompatibility between OpenType and Type 1 (where the glyph widths are expressed accurately as a ratio of two integers using the `div` command).

Unification of accent heights helps to simplify font production and also to make verification easier. An approach to implementing the unified alignment of optical height for all accents common for all lower case accented letters and also for upper case letters (to another vertical level) is very convenient. Fig. 4 shows a sample test, performed also for all text fonts with a subset of accented letters used in Czech and Slovak, with satisfactory results. The base lines, the boundaries of hinting zones or other vertical levels are marked by the auxiliary horizontal lines.

7 Accents: positioning and shapes

Because LM and CS are both descendants of CM, the glyph images of the common character set are expected to be the same or the differences should be (and in fact are) minimal. Therefore, we concentrated our attention on the accents.

Figures 5 and 6 study several selected accented letters from LM and CS in more detail. In some cases `lmbx10` from LM (stroked outlines) looks better than CS (filled bitmap). Most notably, the uppercase accents in CS, designed in METAFONT more than 15 years ago for dot-matrix printers or used at 300 dpi resolution with laser printers, touch or even overlap

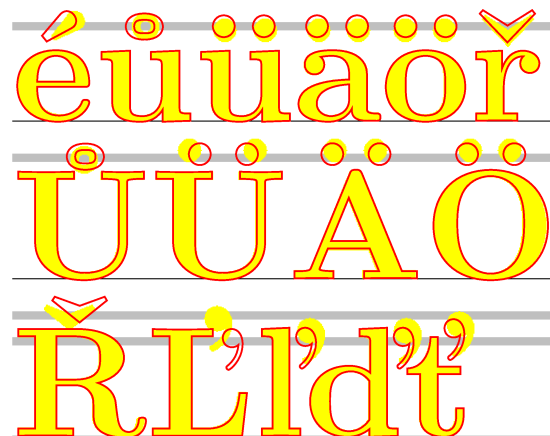


Figure 5: `csbx10` (bitmap) vs. `lmbx10` (stroked outline).



Figure 6: Additional glyphs, comparing `csbx10` and `lmbx10`.

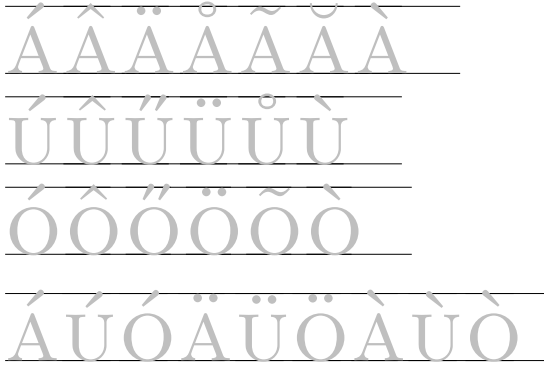


Figure 7: Accents in lmcsc10 (lower case SmallCaps).

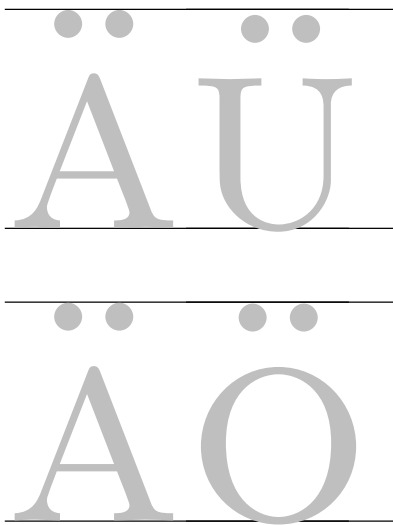


Figure 8: Violation of unified vertical positioning (lower case SmallCaps).

the letter areas. In the current version of LM, probably only \check{R} needs to adjust the horizontal positioning of the caron.

More complex is the situation with “special” typefaces. For the lower case small capitals (lmcsc10 and its oblique variant) in Fig. 7, with the enlarged detail in Fig. 8, the vertical accent levels are consistent for one letter but are not identical across letters, e.g. the lower case U has evidently had its accents lowered.

And for both SmallCaps and Dunhill (lmdunh10 shown in Fig. 9), I think, the unified optical (lower case) accent level is not the best solution. Accents such as ¨ and ˜ seem to be located too high and the gaps between these narrow (in vertical direction) accents and letters look too big. Additionally, the lower case ring accent in lmdunh10 (and lmduno10) is located higher than other accents.

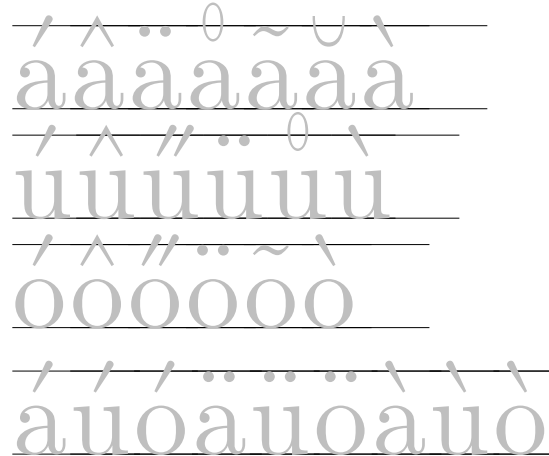


Figure 9: Accents in lmdunh10.

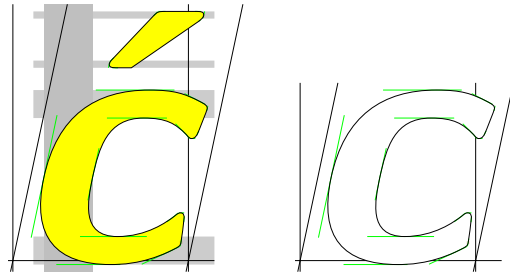


Figure 10: Extrema points and hints in lmsbo10.

8 Comments on outline font representation

Several glyphs do not fulfill the strongest criteria for the best preciseness or conciseness, and still have tiny defects or small inconsistencies. Their tuning or improvement may be discussed; however, it cannot be considered critical.

Figure 10 demonstrates a few aspects of the font design in LM. In the left part of the picture the technique of hinting the accents is presented. It is widely used in LM and I consider this approach convenient.

The oblique fonts (like lmsbo10) have been derived from their upright origins and are the result of conversion and transformation. The outline representation is faithful and “optimal”, i.e. it fulfills the “conciseness” consideration. We use fewer Bézier curve segments but, on the other hand, the redundant points at extremes are omitted and the vertical hints look strange or atypical — the boundaries of the hinting zones do not fit in (missing) extrema points. I am interested in comments about this situation.

9 What the tests do not cover

Verification of metrics has been restricted to T1 (EC) encoding (and compared with CM and CS metric data): character widths together with kerning and ligature pairs. Testing of “internal” font matters was foremost; ordinary tests of typesetting real text was not the main goal.

10 Topics for further discussion and conclusion

In the conclusion we summarize some problems that remain to be solved:

- Accent positioning in special cases: Small caps or Dunhill fonts (Fig. 7 and 9).
- Large number of (irrelevant) kerning pairs in metrics (Tables 4 and 5).
- checking of other metric data, e.g. character heights and italic corrections, which are important for typesetting math.
- Points of extrema and hints in (derived) oblique typefaces (Fig. 10).
- tuning of small details in glyph representation.
- proposals for further tests and other improvements (nothing is absolutely perfect).

The Latin Modern fonts fulfill a high quality of technical realization; Type 1 versions are generated by MetaType1 properly, and contain a minimal number of bugs. Remaining tiny defects in online approximation have no practical influence to the printed output of final documents. The metrics (i.e. character widths and kerning pairs) for the CS subset of the T1 (EC) encoding are acceptable; accents and their placement are also acceptable (in most cases).

The LM text fonts could be taken as finished; the LM math fonts in OpenType are (probably) still under development. I have not checked L^AT_EX support or encodings other than T1 (EC). I expect the L^AT_EX users of LM are and will be testing L^AT_EX, dvips, pdfT_EX and other packages during their exploration of LM together with new additions (e.g. new T_EX metrics).

Generally, we can be satisfied with the text LM fonts in the version 1.010(x), and thank the authors for their successful work(s) and wish them further success in the future.

References

- [1] Donald E. Knuth. *The METAFONTbook*. Addison-Wesley, 1986. Volume C of *Computers and Typesetting*.
- [2] Donald E. Knuth. *Computer Modern Typefaces*. Addison-Wesley, 1986. Volume E of *Computers and Typesetting*.
- [3] Computer Modern fonts. CTAN:/fonts/cm.
- [4] CS fonts. ftp://math.feld.cvut.cz/pub/cstex/base/csfonts.tar.gz.
- [5] EC fonts. CTAN:/fonts/ec.
- [6] Latin Modern fonts. CTAN:/fonts/lm.
- [7] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating. *Proceedings of EuroT_EX 2003*, Brest, France, 24–27 June 2003. *TUGboat* 24:3, pp. 575–581.
- [8] Bogusław Jackowski, Janusz M. Nowacki. Enhancing Computer Modern with accents, accents, accents. *TUGboat* 24:1, 2003.
- [9] Bogusław Jackowski, Janusz M. Nowacki. Latin Modern fonts: How less means more. *Proceedings of the XV EuroT_EX 2005 conference*, Pont-à-Mousson, France, March 7–11, 2005.
- [10] Bogusław Jackowski, Janusz M. Nowacki. Rodzina fontów Latin Modern. *Biuletyn Polskiej Grupy Użytkowników Systemu T_EX*, Zeszyt 23:9–12, 2006.
- [11] Karel Piška. Font verification and comparison in examples, *EuroT_EX 2006 Proceedings*, *TUGboat* 27:1, pp. 71–75, 2006.
- [12] Karel Piška. Procedures for font comparison, 2007. *Proceedings of the EuroT_EX 2007 conference*, *TUGboat* 29:1, pp. 50–56, 2007.
- [13] Will Robertson. An exploration of the Latin Modern fonts. *TUGboat* 28:2, pp. 177–180, 2007.

The GUST Font License: An application of the L^AT_EX Project Public License

Jerzy Ludwichowski
Nicholas Copernicus University
Toruń, Poland
Jerzy dot Ludwichowski (at) uni dot torun dot pl

Karl Berry
T_EX Users Group
karl (at) tug dot org

Abstract

We describe interpretation and usage of the GUST Font License. It is legally identical to the L^AT_EX Project Public License, with additional requests (not requirements) related to usage with fonts. It is currently used for the Latin Modern font project, among others.

1 Introduction

A previous article [9] described the process of formulating a license for fonts in the T_EX world, culminating in two different licenses, based on (but slightly modified from) the L^AT_EX Project Public License (LPPL) [8]. These were the GUST SOURCE and GUST NOSOURCE font licenses. After further discussions, we realized that the licenses could be simplified and combined into one; viz., the GUST Font License (GFL). Furthermore, it could be made legally identical to the LPPL.

In [9], we also considered the GNU General Public License [1] and the SIL Open Font License [10], among others, before settling on the LPPL. We won't repeat that analysis here. The present article describes interpretation and usage of the new license and points out some benefits of the new formulation.

2 LPPL interpretations

The main reason for the two separate licenses in our first attempt was the fact that fonts do not always have source files separate from the fonts themselves; they can be designed purely visually.

We owe a large debt to Frank Mittelbach of the L^AT_EX team, and the principal architect of the LPPL, for pointing out that the LPPL does *not* require that source files exist. In LPPL terms, the 'Work' and 'Compiled Work' can be one and the same thing, which is indeed the case for visually defined fonts.

A second concern in the original formulation was clause 6a in LPPL, which requires, in certain cases, that a derived work which can directly replace an original work identify itself as a modified

version. It was not clear to us how this could apply to fonts. Fortunately, Mittelbach again disentangled us, noting that simply changing the font's official name (e.g., the `FontName` in a Type 1 PostScript font) would suffice to fulfill this clause.

With these clarifications, we realized that the two GUST font licenses could be combined into one, and furthermore, the result could be made legally equivalent to the LPPL—we would only need to add requests, without any new requirements or other changes. This is highly beneficial, as the FSF, Debian, and other organizations have already officially accepted the LPPL as a free software license [3]; this way, there would be no need for additional analysis, and no question that the new GUST Font License would also be a free software license. We were also pleased not to contribute to the ongoing problem of "license proliferation" in the free software world [2]. Finally, of course the LPPL was designed for use with (L^A)T_EX, so we were very happy that it could be used for fonts in the T_EX world, too.

3 GFL usage

As a result of the above, to use the GFL, strictly speaking, it is only necessary to abide by the LPPL. Most importantly, the LPPL maintenance status and any maintainer(s) should be stated. Indeed, this feature of the LPPL was a primary reason for choosing the LPPL in the first place, as explained in the prior article [9].

The additional request in the GFL is for names of fonts to be changed in derived works, at the authors' option. The recommendation thus is for authors to provide a separate manifest file, with three sections:

1. Font names which should be changed.
2. File names which should be changed.
3. File names which need not be changed.

The Latin Modern manifest [6] is a good illustration.

The GFL web site [4] provides a generic template for a manifest file in GFL-licensed distributions, as well as a template for a readme file, and of course the current text of the GFL, among other information.

GUST e-foundry fonts [5] will be released under the GFL as new versions are published; Latin Modern has been already, as mentioned above. We are also pleased to report that Palle Jørgensen has agreed to use the GFL for his Fonetika Dania [7].

4 Conclusion

We hope the GFL will remain a stable legal basis for font releases in the T_EX world for many years. Questions and comments are welcome, as always; please see the GFL web page [4] for contact information.

Finally, thanks again to Frank Mittelbach for his time and good cheer in discussing these perennial licensing issues.

References

- [1] GNU General Public license. <http://gnu.org/licenses/gpl.html>.
- [2] GNU Project. Licensing web page. <http://gnu.org/licenses/>.
- [3] GNU Project. Various licenses and comments about them. <http://gnu.org/licenses/license-list.html>.
- [4] GUST Font License web page. <http://www.gust.org.pl/fonts/licenses>.
- [5] GUST e-foundry web page. <http://www.gust.org.pl/projects/e-foundry>.
- [6] Jackowski, B. and Nowacki, J. M. Latin Modern manifest. <http://ctan.org/tex-archive/fonts/lm/doc/fonts/lm/MANIFEST.txt>.
- [7] Jørgensen, Palle. Fonetika Dania. <http://ctan.org/tex-archive/fonts/fonetika/>.
- [8] L^AT_EX Project Public License. <http://www.latex-project.org/lppl/>.
- [9] Ludwichowski, J. GUST font licenses. *GUST Bulletin 23*. <http://www.gust.org.pl/projects/fonts/licenses/gfl.pdf>.
- [10] SIL Open Font License. <http://scripts.sil.org/OFL>.

A brief history of T_EX, volume II

Arthur Reutenauer*

ENST Bretagne

Technopôle Brest-Iroise

CS 83818

29238 BREST CEDEX 3

arthur dot reutenauer (at) normalesup dot org

Rationale for this *volume II*

When I gave this talk in Bachotek I appended the subtitle *Pax T_EXnica — the program on which the sun never sets* — an obvious pun to two historical empires renowned for their considerable geographical extent. I wanted to add this subtitle for two reasons: first, I liked *A brief history of T_EX* a lot but I realized after choosing it that there had already been a talk with this exact same title more than ten years before¹ and I wanted to avoid the risk of confusion, be it only for archival purposes; and second, I felt the subtitle made my standpoint clear — the history I wanted to account for was very much a *geographical* one, namely how T_EX enabled us to gradually typeset in every language of the world — or almost so. As far as the printed version was concerned, though, it seemed that it could also be considered a sequel to the first article — after all, many things have changed over ten years! Hence this *volume II*.

But first, let us recapitulate things from the beginning ...

1 The origins

1.1 In the beginning there was ...

History begins, scholars tell us, with the invention of writing and the ability to account for one's own culture. So, in the beginning there was typesetting and the program that enabled us to do so, let us call it T_EX. This program was written by a man, let us call him 高德纳. Oh, and we need a date, too, so let's say 1978, thirty years ago.

高德纳, as the name suggests, lived in a region inhabited by many Chinese citizens, next to the great city that goes by the name of the Old Golden Hills. But he was an American citizen and a native speaker of the English language. So the program he wrote

* I wish to thank Jerzy Ludwichowski heartily for his constant encouragement to write this article and his patience in waiting for it.

¹ This talk was given in Toruń in 1995 and published the next year by different journals, including *TUGboat*, where it is available online: <http://www.tug.org/TUGboat/Articles/tb17-4/tb53tay1.pdf>.



Figure 1: The name of a distant galaxy. From the very beginning, T_EX sets out to conquer the universe (extract from `story.tex`, in *The T_EXbook*, chapter 6).

was all in English (with a lot of ‘\’ though) and it was meant — at first — for English speakers to use.

Nevertheless, when 高德纳 created T_EX, he still thought of the users speaking other languages. Of course, all the commands were in English, the default settings were chosen for that language, and the fonts used a 7-bit encoding² supporting only the Latin script, but he made provisions for extending this. The fonts, in particular, were supplied with a set of diacritics with the help of which he devised a set of well-known accent commands that could construct an accented character “on the fly”, a sample of which can be seen in figure 1, an extract of a famous T_EX file. This made it possible to write, mostly, all the languages of Western Europe — and therefore of all the other countries that use the same languages, in particular all of South America.

This was the first step since, even if they may seem impractical now, the accent commands actually introduced a way of inputting a lot of characters users didn't have access to on a standard American keyboard, much in the way math commands were a way of specifying the layout of complicated math formulæ; so even if they were not an encoding³ in the current meaning of the term, they were a sort of coding system, and were thought as such by many users as well as some recoding utilities.⁴

So T_EX extended, from the very beginning, over all the Americas as well as the western part of Europe,

² That is, they could only have up to $2^7 = 128$ characters.

³ In the sense of an encoded character set, like ASCII, the ISO-8859-* family, or Unicode.

⁴ To name just one, the very popular `recode` program (`ftp://ftp.gnu.org/pub/gnu/recode/`) knows about sequences like ‘e’ as the “T_EX” encoding.

“Uznając, iż los nas wszystkich od ugruntowania i wydoskonalenia konstytucji narodowej jedynie zawisł, długim doświadczeniem poznawszy zadawnione rządu naszego wady, a chcąc korzystać z pory, w jakiej się Europa znajduje i z tej dogorywającej chwili . . .”

Figure 2: Polish uses a lot of diacritics (extract from the May Third Constitution, 1791).

and many regions in Africa.⁵

But, as mentioned, this wasn’t enough even for some other languages using the Latin alphabet, let alone languages using any other alphabet or different writing systems. Work needed to be done, as 高德纳 acknowledged that he couldn’t handle all the languages of the world by himself, and he encouraged people to settle to this task. It wasn’t long before people did so.

1.2 Go East

As T_EX was born, the story tells us further, there was a companion program called METAFONT, whose purpose was to design the fonts that T_EX used. As a matter of fact, all the letters and accents we discussed above were all drawn using METAFONT, so adapting the fonts to other languages meant, mostly, drawing more characters as needed.

Let’s discuss how this was done. An interesting example is Polish. It uses a wealth of accents (see figure 2); most of them were already present in the fonts or easy to add, by simple modifications to the existing characters. One of these accents, though, is quite special: it’s called *ogonek* which means “little tail” in Polish, as for example on the first word of figure 2. It looks remotely like a reversed cedilla but is not quite, and the drawing had therefore to be invented from scratch and polished carefully.⁶ Then, a new control sequence had to be invented and agreed upon in order for users to be able to input *ogonek*-accented characters, in the same spirit as the already existing accent commands; nowadays, it’s `\k` in L^AT_EX.⁷

⁵ Including, of course, all the languages of the former colonizers like English and French, but also important African languages like Swahili which are written entirely in the Latin alphabet.

⁶ The Poles are very proud of their *ogonek* and you should not upset them by speaking ill of it. Maybe it is even too daring in the eyes of some to state that *ogonek* resembles a reversed cedilla!

⁷ For a thrilling account of how T_EX came to Poland, I highly recommend reading the text of this talk, given at the TUG meeting in Hawai’i in 2003, and published in *TUGboat*, volume 24, number 1: <http://www.tug.org/TUGboat/Articles/tb24-1/odnyniec.pdf>.

Over the years, more characters were designed and entire alphabets were digitized using METAFONT, starting with Greek and Cyrillic, which were drawn by various people around the world.

An important step was when T_EX was extended in 1989 to handle 8-bit input (then becoming T_EX3), thus enabling fonts to have up to 256 characters. The next year, during a meeting in Cork, T_EX users from all over the world agreed on a standard encoding for T_EX’s Latin fonts, which then came to bear the name of its birth place (or the alternative, less poetical names of T1 and 8t). Another important milestone at that time was the advent of the L^AT_EX *babel* package, which attempted to provide a convenient way to switch between languages and a common interface for L^AT_EX users.

But even after those fonts were designed, after those standards were agreed upon, many things were left to do: what about Arabic, for example? T_EX offered amazing possibilities, but did not really address the issues of right-to-left typesetting and it also completely left aside the fact that characters can have different forms according to their place in a word (both being essential features of Arabic). Therefore, to go further it was necessary to *think different!*⁸

2 Think different

2.1 T_EX encompasses the *Mare Nostrum*

As early as 1987, the first experiments were made in handling the challenge of Arabic typesetting and gave birth to a modified version of T_EX called T_EX-X_FΓ, to emphasize the fact that it could write in two different directions.⁹ This worked in a particular way: when writing data in the output file, T_EX-X_FΓ did not reverse the order of letters but wrote a mark whenever it encountered a sequence of Arabic letters, and then let all the work be done by the printer driver. That way, the text was stored in *natural* order in the output file — that is in the order in which an Arabic speaker would speak out the letters — but, on the other hand, it meant the files output by T_EX-X_FΓ had to be processed by a special driver. So 高德纳 — who, once again, was the lead in that project — decided that the output format

⁸ To quote an old slogan of one of the big computer manufacturers — I’m not sure what the legal status of such commercial slogans is and I may not be entitled to reuse it in a document; but I want to make sure people know I didn’t mean any harm and in case TUG is sued I deny everything.

⁹ The founding article was published in *TUGboat*, volume 8, number 1: <http://www.tug.org/TUGboat/Articles/tb08-1/tb17knutmix.pdf> and makes fascinating reading even today, especially when compared with the current paradigm established by Unicode in that area — the so-called bidirectional algorithm.

input: Book is "باتك" in Arabic
output: Book is "كباب" in Arabic

Figure 3: The challenge of Arabic writing: when setting an Arabic text, the order of the letters does not only need to be reversed, but their shapes also may vary a lot — can you recognize all four of them on the second line?

should be called DVI-IVD, to differentiate it from the traditional DVI output format.

In this way, both the \TeX program and the DVI format were “extended” in the sense that they were made able to handle different types of information in addition to the ones they already knew how to process or store. We shall meet a lot of these along the way, and we shall refer to them as “extensions” — or sometimes “engines” for \TeX extensions. So \TeX - \XeTeX was, probably, the very first \TeX extension.

A few years later, \TeX - \XeTeX was itself extended via something that achieved roughly the same goals, but without needing to resort to an extension of the DVI: it readily reversed the order of each letter in the output file as appropriate. To mark both the similarities and the difference of this second extension with the first one, it was called by the same but a second hyphen: \TeX -- \XeTeX !

These improvements were interesting and made Arabic typesetting with \TeX possible quite early; but it was still an experimental system, and apart from that, it did not change things for other scripts such as, in particular, the Indic and South Asian scripts.

2.2 Enter Unicode

For better-suited treatment of such complex scripts, Omega¹⁰ was designed. It consisted of several major improvements:

- It enabled (probably) every sort of writing directions.
- It came with a set of filters (the Ω transformation processes, Ω TP for short) that transformed the input text.
- It enhanced the traditional font formats used by \TeX from 8-bit-based encoding to 16 bits.

The two first points made the treatment of Arabic much more natural (just switch the writing direction from left-to-right to right-to-left, top-to-bottom; and filter the input text to give each letter its appropriate appearance given the context); and the third one was also very important because it addressed the problem which we haven’t yet mentioned: up to then, \TeX handled only fonts with at most 256 slots.

¹⁰ We shall call it simply by the Greek letter from now on.

This isn’t so important for alphabetic scripts, but becomes a major issue when one wanted to typeset in a language using ideographs, whose number by far exceeds this limit.

Ω addressed part of this problem by making direct use of (possibly) very large font metrics; that is, it could use any font on the input but remained constrained by the output format.

Anyway, it brought with it a conceptual leap, even if it failed to address some of the issues of the output format. Over the years it has been successfully used to typeset the Devanāgarī, Malayalam, Tibetan, Inuktitut and Cherokee scripts among others, although it has never really gained a wide acceptance.

2.3 The other way: Generating PDF

Ω was first formally released in 1994, and by that time there was a document format that was increasingly gaining in popularity and commercial strength: PDF. Hearing about this “Portable Document Format” in the \TeX world, one cannot help thinking that it is a concept quite close to the traditional output format, DVI (does it not stand for “DeVice-Independent?”); therefore it seemed only right that \TeX should be able to produce PDF directly: and so it did, with the birth of the well-known pdf \TeX on March 15th, 1997 (then under the name `tex2pdf`).

Another huge improvement pdf \TeX brought was the direct handling of TrueType fonts, which by that time had become a major font format for personal computers.

2.4 One more ε extension . . .

Worth mentioning here, since its later development was to be closely related to pdf \TeX ’s, is “the” extension of \TeX , called ε - \TeX for “extended \TeX ”. Developed during the late 90s, it extended the above-mentioned \TeX -- \XeTeX (the second one, with two hyphens) and was therefore of great use to Arabists and other communities writing from right to left.

Some time later, its very useful extended features were merged into pdf \TeX , which thus had for a while an offspring called pdf ε - \TeX , now fully incorporated into pdf \TeX ; that is, pdf \TeX now supports the ε - \TeX extensions, but it can also pretend to know nothing about these and be simply pdf \TeX .¹¹

2.5 Needless To Say

Before proceeding to the last part of this account, there are a few words to be said on another attempt

¹¹ Just as it also could, from the very beginning, behave as the DVI-producing \TeX , or the actual pdf \TeX — which means that in DVI mode and *with* the extensions . . . you probably get the picture.

of extending T_EX, which isn't very famous now but whose name still lingers in many memories. It was to be a completely new concept, opening up a world of possibilities . . . but as of today, it is no more.

The “New Typesetting System”, as it was called — or $\mathcal{N}\mathcal{T}\mathcal{S}$ for short — was a complete reimplementa- tion of T_EX in Java, aiming at full compatibility with the original engine, and providing at the same time the great modularity and extensibility that comes with that language.

Sadly, while the first goal was actually achieved (T_EX was indeed rewritten in Java), it proved completely unusable and pointless because of its extreme slowness. The extension projects were never carried out and $\mathcal{N}\mathcal{T}\mathcal{S}$ has now been officially declared dead.

3 Rule, T_EXannia, T_EXannia, rule the waves

3.1 Taming the multilingual lion

Back to living projects now: there is one very young lion which has brought many changes for a lot of users recently. X_ƎT_EX, as it is called, is an extremely multilingual extension of T_EX; the very name suggests, again, that it can typeset in every direction (it can spell “T_EX” backward). Its spirit is a bit special in that it started off (in April 2004) as a Mac OS-specific program which made heavy use of the Apple libraries designed to handle text and scripts.¹²

Shortly thereafter (April 29th, 2006), X_ƎT_EX was released for Linux too, and it was not long before it was ported to Windows as well.

X_ƎT_EX's main distinction — and its overwhelming advantage for many of its adherents — is to get rid of nearly all the hassle in font selection, font installation, etc., while opening up at the same time a whole new world of possibilities: people can suddenly use the bleeding-edge features of the newest font technologies with no particular problem. The key to this was the use of a lot of external libraries — which of course comes at a price: users lose part of the control over every detail of the processing chain which had always been a great advantage of T_EX; but many find this tradeoff acceptable.

3.2 Towards the infinite and beyond

Another T_EX engine worth mentioning in passing is called Aleph (\aleph), of Ω -ish ascent. It started as an attempt to stabilize Ω while merging the extensions by ε -T_EX at the same time (hence its original name, ε - Ω).

¹² Before X_ƎT_EX there was T_EXGX (on the Mac only) which used the same series of Apple libraries, then called “GX technology” — TrueType GX was an extension of the TrueType font format, now replaced and enhanced by AAT — Apple Advanced Typography.

While it attracted much attention for a few years after it was launched in 2001, it is today overshadowed by another successor to T_EX, which is now thought of as representing the future path.

4 Howling to the moon

This “successor” is LuaT_EX. As this seems to be yet another prefixed version of T_EX, we shall first explain what that prefix is. Lua is a small scripting language originated at a university in Rio de Janeiro (Brazil) which was developed to be embedded in other applications. The word “lua” means moon in Portuguese (hence the title of this section).

So the idea seems clear, LuaT_EX is Lua + T_EX: an embedded language in T_EX, enabling us to go even further than anything that could be done before with macros; in LuaT_EX we will also have the Lua language and we can write Lua functions in addition to T_EX macros. And . . . there is actually much more: while LuaT_EX was indeed meant to be Lua + T_EX (actually pdfT_EX, now merged into pdf ε -T_EX) when it was first conceived in the beginning of 2005, it is now also incorporating the features of \aleph and its parent Ω , therefore effectively merging two families of engines: the “ Ω way” and the “pdfT_EX way”. Lua will be present at every stage of the processing chain, with *callbacks* enabling the user to redefine parts of T_EX's tasks using Lua functions. Finally, METAPOST is planned to be part of it too, being rewritten as a library (instead of a stand-alone program).

LuaT_EX is under active development today and a first public release is planned for the 2008 TUG conference in Ireland.¹³

5 Back to the future

With LuaT_EX we have reached the most recent developments in T_EX, and here it seems nice to say some words to summarize the changes that we have seen above.

If any general view is to be had, it seems to me that the main changes that T_EX has undergone over the years were not only major improvements but genuine *Copernican Revolutions* which progressively widened T_EX's field of application. I have tried to classify those phases in the article by making each one of them a different section: section 1 shows how T_EX started with an approach of typesetting akin to that of the craftsman's carefully setting type to build a page of text,¹⁴ while undergoing an initial modest expansion along with some “standardization” (Babel

¹³ When TUG will return to Cork, which became famous in the T_EX world 18 years ago!

¹⁴ Let us not forget: “Rhymes are typeset with boxes and glue”, in *The T_EXbook*, chapter 14.

package for \LaTeX , Cork encoding). Then it went through a phase where the first experiments were made to handle “complex scripts” (section 2) and this gave birth to the first true extensions of \TeX which are actually quite old (again, \TeX -- $X_{\text{E}}\Gamma$ was written 20 years ago). These extensions were consolidated in the recent past described in section 3, when \TeX showed how it could still keep up with major changes in the printing industry (PDF, TrueType and then OpenType fonts). The present of \TeX development (section 4) is exemplified by $\text{Lua}\TeX$ which, once again, comes with a complete change of perspective on \TeX processing. These have been the four “ages” of \TeX .

Writing history is important, and I have always got the impression that the \TeX community did not care enough about its own history — there are of course well-known bits and anecdotes about \TeX , but those are closer to legend than to history. Writing history is also a difficult and humble task, and I cannot claim having covered everything that was worthy remembering about \TeX : some words could have been said about macro packages (beyond \LaTeX , which I simply quoted in passing) as well as \TeX distributions which have contributed a lot to \TeX ’s expansion. Nor have I talked about important “industry standards” such as XML which have also become an important part of \TeX ’s capability today (this would have been more linked to the macro packages than to the engines themselves). It is therefore my hope that we can, maybe, enhance this article with more descriptions and memories, and I have opened a small page at the $\text{Con}\TeX\text{t}$ wiki to discuss this: http://wiki.contextgarden.net/History_of_TeX.

To conclude, I would like to name a few places where I’ve personally encountered \TeX , as it gives an idea of the versatility of \TeX and the extent of the *Pax TeXnica*:

- The general scientific community and especially mathematicians and computer scientists.¹⁵
- People from humanities, especially in Ancient Greek and linguistics.¹⁶

¹⁵ This was obvious but I felt I still had to mention it first!

¹⁶ Indeed, what other free program can handle at the same time Ancient Greek, Russian, Lithuanian, Latvian, Sanskrit and French? Someone at my university was doing a Master on Indo-European linguistics and did really need to input all these languages.

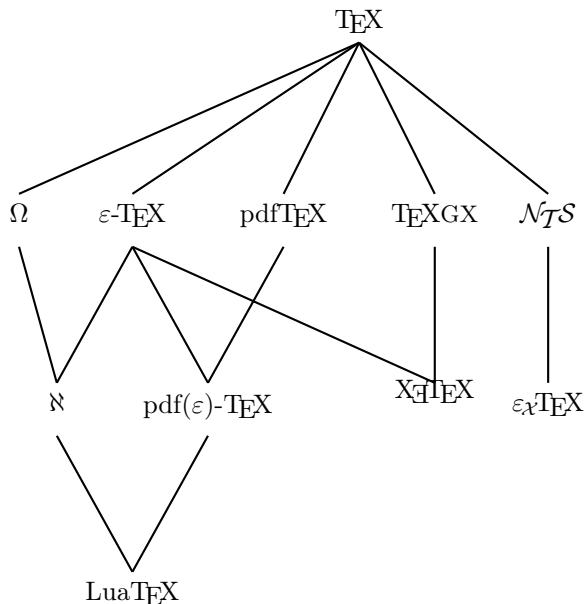


Figure 4: The happy \TeX family. The different extensions have been divided into successive “generations” of engine, corresponding to the different sections in this article.

- Shopkeeper in one of the biggest Chinese bookshops in Paris.¹⁷
- Musicians needing to engrave scores.¹⁸
- Users of free software.¹⁹
- People involved in the publishing industry.²⁰

¹⁷ Perhaps my most amazing encounter with \TeX in a place I didn’t expect it at all, but I swear it is true: while gazing at the shelves of the aforementioned bookshop I overheard two members of the staff discussing how to produce documents in Chinese (probably for the shop’s catalog).

¹⁸ $\text{Music}\TeX$ and $\text{Musix}\TeX$ have many an adept in spite of their extreme difficulty to master.

¹⁹ Indeed, on an average Linux distribution, there are very little software able to rival \TeX — OpenOffice is an obvious example, but it may be the only other one.

²⁰ Especially for processing XML, as already mentioned.

Overview of the T_EX historic archive

Ulrik Vieth

Vaihinger Straße 69

70567 Stuttgart

Germany

ulrik dot vieth (at) arcort dot de

Abstract

This paper presents an overview of the T_EX historic archive, an archive of historic T_EX distributions and packages hosted on the TUG FTP server. While the topic of this conference focuses on paths to the future, this archive aims to preserve snapshots of significant past and present releases of T_EX software, so they will remain available for reference and won't get lost when new versions are installed. Apart from explaining the motivation behind the project, this paper also presents an overview of what the archive currently contains and what is still missing.

1 Introduction

The history of T_EX goes back nearly 30 years now, starting from the earliest beginnings in 1977. In the case of T_EX, unlike most other software projects, the early history is pretty well documented in the literature [1, 2, 3], at least as far as the core system of the T_EX engine itself is concerned.

When it comes to related elements of the T_EX system, such as macro packages or fonts or systems like METAFONT or METAPOST, the documentation of the history isn't nearly as sophisticated as in the case of T_EX, and pretty soon one ends up piecing together the history from anecdotal evidence.

Probably the best way of preserving the history of a software project is by preserving the artifacts of the system, preferably the original sources and distribution packages of all significant releases.

While this may seem to be a well-established best practice in the age of project web sites featuring source code repositories and distribution archives, one has to keep in mind that such a level of project infrastructure is quite a recent achievement, far from standard just a few years ago.

2 Genesis of the T_EX historic archive

Looking back just one decade, establishing CTAN as a comprehensive archive network of all available T_EX software was a great achievement of the combined T_EX users groups, which has become a resource that is taken for granted now.

From the point of view of software archaeology, however, such an archive doesn't necessarily satisfy all the needs, since the different types of archives have different goals and different audiences.

While CTAN is a dynamic and ever-changing archive, which aims to carry only the latest versions of everything available, a historic archive is a static archive, which aims to preserve archival copies of various release versions of selected packages which may be of specific historic interest.

While CTAN aims to satisfy the needs of end users, making it easy to find the current version of a package without causing confusion, the audience of a historic archive is likely to consist of just a small group of expert users, who might be interested in browsing and comparing various different versions of a package with specific attention to small details such as original file dates and timestamps.

Based on these considerations, the idea of an archive of historic T_EX distributions and packages was conceived in the late 1990s in discussions between the author and some CTAN maintainers.

It was agreed that the historic archive should be kept separate from CTAN and it was decided to establish the archive on the TUG FTP server, which also served as a development server for some projects and already carried some interesting material.

3 Organization of the T_EX historic archive

As explained, the T_EX historic archive is intended as an archive of historic T_EX distributions, packages and files, which is hosted on the TUG FTP server at the following URL:

`ftp://ftp.tug.org/historic/`

The archive is maintained entirely manually, mostly by the author himself with a few contributions from others. It is updated infrequently on an occasional basis whenever new material becomes available that may be of significant historic interest.

The organization of the archive loosely follows the organization of CTAN, featuring top-level directory such as `fonts`, `macros`, `systems`, etc.

Below the top level, however, the structure is somewhat different. In some cases, it was sufficient to have a single directory for a package, containing multiple copies of distribution files that can be distinguished by version number. In other cases, it was more convenient to have individual subdirectories for each release of a distribution or package.

4 Contents of the \TeX historic archive

When the \TeX historic archive was first established, it was mostly based on the author's personal archive of material collected during the 1990s while serving as the local \TeX administrator on a Unix network at university and on Linux PCs at home.

Later, some additional material was added that was restored from old CTAN CDs or mirrored from various FTP servers located by search engines that were still carrying old material at that time.

In some cases the most recent versions of some packages were mirrored directly from CTAN or from distribution archives of project web sites.

The selection of material is obviously somewhat biased by personal preferences and interests, so the coverage of Unix-based distributions is quite extensive while the coverage of other platforms is largely missing. Similarly, the coverage of \LaTeX releases is very comprehensive, while \ConTeXt is still missing. Fortunately, there exists an independent archive site of \ConTeXt releases to fill the gap in this case.

In the following sections, we will look at some examples of distributions and packages preserved in the collection of the \TeX historic archive.

systems/knuth This directory contains the true original distribution files from Don Knuth, mirrored from Stanford, covering the 1995, 1998, and 2002 releases of \TeX and METAFONT.

In principle, the material is identical to what is found in `systems/knuth` on CTAN, except that those files are somewhat less reliable, because they are unpacked and rearranged without much regard to preserving timestamps or the original structure.

systems/unix This directory contains various releases of Unix \TeX distributions by Pierre MacKay. The coverage is complete from \TeX 3.0 to \TeX 3.141, but earlier releases are very rare.

The author was lucky to find a complete copy of \TeX 2.95 to mirror from an obscure FTP server during the late 1990s, but he was unable to locate a complete copy of \TeX 2.0 at that time.

Since the Unix \TeX distribution used to be quite popular at times, it is quite likely that some copies of distribution tapes of older Unix \TeX releases may have survived in some basements or attics. However, finding a suitable tape drive and getting old tapes to read after 15 or 20 years could be quite difficult. On the other hand, unpacking the files once they are restored to disk should be no problem at all, since the TAR format has remained popular and is still widely used on Unix/Linux platforms today.

systems/web2c This directory contains various releases of the `web2c` distribution by Karl Berry and later Olaf Weber. The coverage is mostly complete from `web2c-5.0` of 1990 to `webc2c-7.5.4` of 2005 and also includes many beta-test releases.

In recent years, maintenance of `web2c` was integrated with \TeX Live and has been moved to a Subversion repository on `svn.tug.org`. While such a source code repository provides even more fine-grained access to changes between releases, the lack of traditional distribution files unfortunately causes a gap in the file-based historic archive.

Moreover, unlike traditional text-based change log files, we are becoming more reliant on the technology of the Subversion repository to preserve the change history. We can only hope that the change history will be properly migrated and not blindly discarded in case the repository is ever moved to another system or platform.

systems/teTeX This directory contains various releases of the `teTeX` distribution by Thomas Esser. The coverage is mostly complete from `teTeX-0.2` of 1994 to `teTeX-3.0` of 2005 with the exception of beta-test releases, such as `teTeX-0.99`.

It is interesting to note how the distribution has grown from the original Linux-only release of `teTeX-0.2` which used to fit on 5 floppy disks to the later multi-platform Unix distributions featuring an ever-growing `texmf` tree of 30, 50 or 90 MB.

systems/metapost This directory features a collection of METAPOST releases by John Hobby and later Taco Hoekwater. The coverage is complete from version 0.62 of 1995 to the latest releases and also includes many beta-test releases.

All of the recent beta-test and production releases were mirrored directly from the project web sites at `sarovar.org` and `foundry.suppelec.fr`, so in this case the flow of file-based distribution snapshots for archival purposes has fortunately remained intact after the development infrastructure was moved to a different platform.

systems/pdftex This directory features a collection of pdfT_EX releases by Hàn Th^é Thành and later Martin Schröder. Unfortunately, the coverage of the archive is currently only partly complete. It is fully complete only for the recent releases from versions 1.20 to 1.40, that could be mirrored directly from the project web site at sarovar.org, but most of the earlier versions are still missing.

The author has tried to fill in some of the gaps by adding versions of pdfT_EX that were bundled with web2c releases, but the earliest versions don't go back any further than pdftex-0.11.

If anybody could provide additional copies of early versions of pdfT_EX, such a contribution to the historic archive would be very welcome.

macros/latex209 This directory contains various releases of L^AT_EX 2.09 of 1991–92, including the final public release that can still be found on CTAN in the `obsolete` tree. It also contains several noteworthy add-on packages for L^AT_EX 2.09 such as NFSS and the so-called Mainz packages that later became part of the L^AT_EX 2_ε kernel or `tools` bundle.

It would be interesting to add original copies of L^AT_EX 2.09 of 1986 or even L^AT_EX 2.00 of 1984 for comparison, if such versions could be recovered.

macros/latex2e This directory features a collection of L^AT_EX 2_ε releases, starting from the earliest pre-test releases of 1993 until the current release. The coverage is very complete for most of the 1990s, but the author has failed to preserve snapshots of some of the more recent releases, which however could be recovered from CTAN CDs.

In addition to the official L^AT_EX 2_ε releases, the collection also includes some alpha-test releases from the author's personal collection that were originally circulated only to a limited group of testers.

Of course, the historic archive cannot possibly cover all the many contributed add-on packages and classes for L^AT_EX, so the coverage concentrates on the L^AT_EX base system and the required packages, but even those aren't fully complete for all releases. The archive usually includes the `graphics` and `tools` bundles, but sometimes lacks releases of `amslatex`, `babel`, or `psnfss`.

macros/context As mentioned before, the selection of material currently doesn't include much in the area of macro packages besides L^AT_EX.

In the case of ConT_EXt, there fortunately exists an independent project web site of ConT_EXt releases since 1997 at foundry.supelec.fr, which could be easily copied or mirrored, if desired.

fonts/cm This directory contains various releases of Computer Modern fonts of 1995, 1998, and 2002, extracted from the corresponding original sources in `systems/knuth`.

In principle, these files are redundant, but they are provided here for the convenience of font developers and researchers, who sometimes need to compare details of various releases of font sources.

It would be interesting to add additional earlier versions of CM fonts for comparison, possibly going back to the first releases of 1985 or even earlier to AM fonts, if such versions could be recovered.

fonts/dc-ec This directory contains various releases of DC and EC fonts by Norbert Schwarz and later Jörg Knappen between 1991 and 1997.

In addition, the collection also includes some test releases from the author's personal collection that were never widely distributed.

fonts/modes This directory features a collection of various releases of `modes.mf` by Pierre MacKay and Karl Berry since 1991, providing a repository of METAFONT mode definitions for various devices.

fonts/fontname This directory contains various releases of the `fontname` document by Karl Berry since 1992, which not only lists the assignment of font names for several suppliers, but also documents several important font encodings.

fonts/utilities/fontinst This directory holds a collection of `fontinst` releases by Alan Jeffrey, Sebastian Rahtz and later Lars Hellström.

The coverage is relatively comprehensive since `fontinst-1.800` of 1998, but only a few copies of earlier releases since 1994 could be recovered from various sources such as CTAN CDs.

Some versions of `fontinst` between 1995 and 1998 never existed as official releases on CTAN and were only available as unofficial releases as part of other font tools, so there remains a gap in the archive that can only be partially closed.

fonts/psfonts/tools This archive complements the collection of `fontinst` releases and contains a collection of font tools that were used between 1995 and 1998 to generate font metrics for `psnfss`, using a combination of Perl scripts and unofficial `fontinst` releases by Sebastian Rahtz.

In recent years, the use of these tools has been discontinued in favor of standard `fontinst-1.9xx` after maintenance of `psnfss` and `psnfss-sources` has been taken over by Walter Schmidt.

fonts/lm Among the most recent additions to the T_EX historic archive are the Latin Modern fonts by the GUST e-foundry.

Unfortunately, the coverage is still fairly incomplete at present. While the development of LM fonts has started in 2002, the author has started taking archival snapshots only recently, when the development was already well advanced, so he missed most of the early development versions.

In order to provide a comprehensive reference for font researchers, it would be interesting to add copies of earlier releases for comparison, if those still exist somewhere. Once again, such a contribution to the archive would be very welcome.

fonts/tex-gyre Finally, the T_EX Gyre fonts by the GUST e-foundry have also been added to the T_EX historic archive, as soon as the first releases have become available.

As a lesson learned from Latin Modern fonts, where taking archival snapshots was started too late, the author has concluded that archiving releases of important projects should better be started as early as possible before anything can get lost when the next release is installed over the previous one.

5 Summary and conclusions

As was explained in the introduction, the historic archive was established for a very specific purpose: provide a collection of archival copies of important T_EX distributions, packages or fonts.

The archive started out in the late 1990s with a small personal collection and has steadily grown over the years, adding new material mirrored from CTAN or project web sites as well as old material restored from earlier CTAN CDs or mirrored from obscure FTP servers still carrying old material.

While the archive covers a lot of ground in many areas, there are still a number of gaps that remain to be filled, not only for older material, but also for some of the more recent development projects.

As was already emphasized, contributions to the archive are very welcome, especially in the areas where gaps remain to be filled, such as in the cases of pdfT_EX and the Latin Modern fonts.

Besides the areas already covered in the archive, there are certainly other areas that have been somewhat neglected so far, but equally well deserve to be archived. Some examples include additional macro

packages besides L^AT_EX, such as ConT_EXt, and additional distributions for other platforms besides Unix/Linux such as emT_EX or fpT_EX. Once again, contributions to the archive are welcome, if you have anything to offer in these areas.

6 Additional resources of historic material

Finally, concerning the early history of T_EX and METAFONT before 1990 there remains the resource of the Stanford SAILDART archive located at

<http://www.saildart.org/>

The SAILDART site hosts an archive of the SAIL computer system at Stanford, restored from backup tapes, which happens to be the very machine used by Don Knuth from the 1970s until 1990, on which T_EX and METAFONT were first developed.

At present the SAILDART archive only allows access to the system areas [TEX,SYS] and [MF,SYS], whereas the private areas [TEX,DEK] and [MF,DEK] remain restricted.

Nevertheless, even without access to the private areas, this archive site provides a wealth of historic material from between 1980 and 1990 that remains largely untapped.

Among the material that may be found here are numerous releases of T_EX and METAFONT sources, numerous versions of CM and AM fonts as well as several early L^AT_EX releases. In addition, one might even find some more exotic stuff here, such as macros for TEX78 or font sources for MF79.

So far, only a few examples have been copied from SAILDART to the T_EX historic archive, such as a snapshot of L^AT_EX 2.0 for T_EX 1.0, but there is certainly a lot more that could be added.

We can conclude that even after 30 years since 1977, the history of T_EX remains an interesting topic of research, considering all those new developments currently going on, as well as all the old material that is left to be rediscovered.

References

- [1] Donald E. Knuth. *The Errors of T_EX*. *Software — Practice and Experience*, 19:607–681, 1989.
- [2] Donald E. Knuth. *Literate Programming*. CSLI Publications, Stanford, CA, USA, 1992.
- [3] Donald E. Knuth. *Digital Typography*. CSLI Publications, Stanford, CA, USA, 1999.

TEX Clinic

Joanna Ludmiła Ryćko

Lichtenrader Str. 42

12049 Berlin

jrycko (at) gust dot org dot pl

<http://typoagrafka.eu>

Abstract

The TEX Clinic is a project, started last year, which takes place at the conference in Bachotek. It's open to anyone who is planning or writing a document and needs assistance with typesetting. Even if “the Patient” doesn't know TEX or L^ATEX very well, our Doctors will help him or her to prepare it or to repair it. The result will be a typographically correct document or at least a prescription for how to meet his or her needs.

1 How does it work?

A group of TEXnicians, willing to help, will show you how to begin or will advise you if you aren't able to go on. The main principle of the Clinic is: you typeset your document by yourself, but you can — anytime — count on the advice and help of the experienced “Doctors”.

2 What do I need?

- Willingness is the first thing you should bring.
- Then you will also need **the text of your document** or at least a small part of it. If you have already started to write it in another program, such as MS Word, we can also help you to convert it to TEX or L^ATEX.
- If you are using a special format, please contact the Team before the conference, to arrange the best way of preparing your work for the conversion.
- It would be good if you could bring your own **computer** (a laptop or even a desktop PC). But if you don't have one or can't bring it to the conference, that won't be a problem. There is a PC room at the conference which everyone can use.

3 Do I have to have TEX installed on my computer?

The answer is: yes and no. Of course it would be better if you had it; we can then start immediately with your questions about your work. But you can also count on our TEXnicians to help with installing TEX on your system on your own during the conference.

We will also have some DVDs with diverse TEX distributions, so you can borrow one from us to install it on your computer.



by Toni Walter

4 TEX? L^ATEX? ConTEXt? Maybe X_YTEX?

Among the various Doctors are experts on every topic with TEX inside, and also some non-TEX topics, such as graphics or Emacs. If you have specific questions, better let us know before the conference, to give the Doctors time to prepare. But mostly there shouldn't be a problem with answering any spontaneous questions on the spot.

If you don't know what is the difference between the TEX flavours mentioned in the head, feel free to ask our Team!

5 Do I have to sign up in advance?

It is not necessary, but would be helpful so we can estimate how many “patients” to expect. See section 7.

6 Does it cost anything?

No, you don't have to pay anything aside from the normal conference fee.

7 Who are the experts?

These are the people, who we know somehow (for example from earlier conferences), who are willing to help others, and who have good knowledge in any topic connected with \TeX or other referring to graphics or typesetting.

Below you will find the current list of our experts. In brackets are the languages spoken by that Doctor.

- Maciej Jan Głowacki – $\mathcal{M}\mathcal{E}\mathcal{X}$ [Polish]
- Jean-Michel Hufflen – $\mathcal{B}\mathcal{I}\mathcal{B}\mathcal{T}\mathcal{E}\mathcal{X}$ [French, German, English]
- Paweł Jackowski – Non-Conventional Methods [Polish, English]
- Jonathan Kew – $\mathcal{X}\mathcal{F}\mathcal{T}\mathcal{E}\mathcal{X}$ [English]
- Jacek Kmiecik – $\mathcal{C}\mathcal{o}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{t}$, $\mathcal{M}\mathcal{e}\mathcal{t}\mathcal{a}\mathcal{P}\mathcal{o}\mathcal{s}\mathcal{t}$, $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ [Polish]
- Adam Kolany – $\mathcal{T}\mathcal{E}\mathcal{X}$, $\mathcal{M}\mathcal{e}\mathcal{t}\mathcal{a}\mathcal{P}\mathcal{o}\mathcal{s}\mathcal{t}$ [Polish]
- Ryszard Kubiak – Psychiatrist with the specialisation Emacsphobia [Polish]
- Krzysztof Leszczyński – $\mathcal{M}\mathcal{E}\mathcal{X}$, $\mathcal{T}\mathcal{E}\mathcal{X}$ [Polish]
- Bogusław Lichoński – graphics, $\mathcal{T}\mathcal{E}\mathcal{X}$ [Polish]
- Jerzy Ludwichowski – General Practitioner [Polish, English, German, Russian]
- Mojca Miklavc – $\mathcal{C}\mathcal{o}\mathcal{T}\mathcal{E}\mathcal{X}\mathcal{t}$ Specialist [Slovenian, English, German]
- Grzegorz Murzynowski – $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, possibly $\mathcal{T}\mathcal{E}\mathcal{X}$ [Polish, English]
- Wojciech Myszka – $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ and pictures [Polish]

- Joanna Ludmiła Ryćko – Co-Ordinator, $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ Diagnostician and Surgeon [Polish, German, English]
- Marek Ryćko – $\mathcal{T}\mathcal{E}\mathcal{X}$ ologist [Polish, English]
- Grzegorz Sapijaszko – $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$, $\mathcal{p}\mathcal{d}\mathcal{f}\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ and all connected with $\mathcal{P}\mathcal{D}\mathcal{F}$ [Polish]
- Andrzej Tomaszewski – typography [Polish]
- Zofia Walczak – $\mathcal{L}\mathcal{A}\mathcal{T}\mathcal{E}\mathcal{X}$ [Polish, English]
- Staszek Wawrykiewicz – Surgeon of $\mathcal{T}\mathcal{E}\mathcal{X}$ implantations [Polish, English]

All our Doctors will have special ID cards at the conference, in a distinguished colour, to make it easier to notice them. On those ID cards we will write their specialisations and spoken languages to help you choose the right one.

How can I contact the clinic doctors?

There is an e-mail address which works 365 days a year; feel free to ask your questions there:

klinika@gust.org.pl

If you have any organisational questions or you want to inform us about your specific needs, please contact me directly:

jrycko@gust.org.pl

Current information on the clinic can be found on the Internet:

<http://www.gust.org.pl/projects/klinika>

Will the Clinic also take place at next year's (2008) $\mathcal{B}\mathcal{a}\mathcal{c}\mathcal{h}\mathcal{o}\mathcal{T}\mathcal{E}\mathcal{X}$ conference?

Of course! There are always people willing to help and another who need the answers to their questions. So if you are asking yourself whether it will be possible to visit the Clinic next time, the answer is “yes”. All you need to do is to prepare yourself and your document.

Parameterized Arabic font development for AlQalam

Ameer M. Sherif, Hossam A. H. Fahmy

Electronics and Communications Department

Faculty of Engineering, Cairo University, Egypt

ameer dot sherif (at) gmail dot com, hfahmy (at) arith dot stanford dot edu

<http://arith.stanford.edu/~hfahmy>

Abstract

We present new approaches to Arabic font development for AlQalam. In order to achieve an output quality close to that of Arabic calligraphers, we try to model the pen nib and the way it is used to draw curves as closely to the ideal as possible using METAFONT. Parameterized fonts are also introduced for a more flexible and dynamic combination of glyphs, to be used in forming ligatures and in drawing whole words as single entities. Quality will improve if words are created as single entities since the Arabic script is cursive. We compare our method to the basic binding of glyphs using simple box and glue mechanisms and also to currently existing font design technologies.

1 Introduction

The process of typesetting languages using the Arabic script is more challenging and more complex than typesetting languages using the Latin script. Previous works [3, 4, 10] indicated the special needs for high quality Arabic typesetting. This paper concentrates on just one special property of the Arabic script, namely being cursive in nature, and presents a way to model this property accurately.

Being a cursive script means that letters interact with each other, and adjacent letters affect each other in many ways. Arabic letters have many forms depending on their position in a word: initial, medial, ending and isolated. Early typesetting systems stored glyphs for each character in each of these different forms and used them when required.

Another feature of the Arabic script is the presence of a large number of ligatures in any text, unlike the Latin script which uses only a few ligatures. Arabic ligatures usually include many letters, sometimes a whole word is one ligature. The longest example the current authors have seen is a ligature of seven consecutive letters. The issue of ligatures is partially solved in contemporary systems by introducing glyphs for only a selected number of letter combinations.

Current font design technologies still treat Arabic glyphs as separate boxes. Advanced technologies like OpenType do allow interaction between different glyphs, through numerous layout features, however, there are still limitations that we discuss below.

Hence, we propose a different solution to give us more quality and typesetting flexibility. Our so-

lution is based on accurately modeling the process of writing Arabic, using the powerful, if underutilized, language of METAFONT.

2 Modeling the calligraphic process

AlQalam was initiated with the intention of typesetting Qur'anic and other traditional Arabic texts. Our goal is to produce an output quality as close as possible to a book written by a calligrapher (the majority of Qur'ans in print today are offset images of hand-written pages). In other words we are targeting the maximum achievable quality and typesetting flexibility. In the past two decades, the approach to typesetting Arabic on computers has been through simplifying the Arabic script for easier modeling. Haralambous [5] discusses the typographical simplifications applied to the Arabic script in these past years. Most of these suggestions for simplification failed over the years to gain any market acceptance. Nowadays, with the existence of more powerful computers and the advances in font technologies, it makes sense to try to model Arabic writing more accurately.

Letters have to be completely interactive with neighboring ones; in fact, an Arabic writer looks at a single word as one entity and all letters in it are drawn accordingly, hence it is like one large ligature. The calligrapher also decides the positioning of the word above the reference line as a single entity, not for each letter alone. Moreover, if the line has a certain horizontal space remaining for one word, the calligrapher will make use of additional ligatures and compress letters together if space is short, or

break some ligatures and extend some letters if extra space is available. Of course there are rules for breaking and forming ligatures and also for extending or compressing letters. Some of these rules have been documented in recent papers written in English [3, 4, 10]. Moreover, it is not acceptable to justify the lines in Arabic only by varying the width of the spaces between words as done in Latin.

We illustrate these ideas with examples scanned from a widespread copy of the Qur'an printed in Al-Madinah [1]. Breaking and forming ligatures is evident in words as أَصْحَابُ becoming أَصْحَبُ , and also أَلْحَجَّ becoming أَلْحَجَّ . Other examples show how the kashida or tatweel (elongation stroke) is used to give words extra length as in أَحَدٌ , أَحَد , and أَحَد . Note that in the latter example, the letter haa' can have different lengths of tatweel, hence it does not make sense to store all these different lengths as glyphs to be substituted when needed.

In some cases, the calligrapher may need to extend more than one letter in a word, for example أَكُونُ extended to أَكُونُ or even أَكُونُ . Notice how the second and third forms are almost 1.5 and 2 times as wide as the first. This property of cursive Arabic script, if made possible in computer typesetting, would allow a higher flexibility in line justification, much more than the unacceptable method of relying only on inter-word spaces.

Completely flexible and dynamic fonts must be available to provide this facility in typesetting programs. When we surveyed the available font design technologies, we concluded that METAFONT is the most suitable. We really need to describe the letters in a very abstract way to make them more flexible, i.e. we need not only design but meta-design the Arabic letters — analogous to the Computer Modern typeface family. By METAFONT, here we mean the language itself and not necessarily the output bitmap formats.

3 The Computer Modern typeface family

The Computer Modern (CM) typeface family produced by Donald Knuth [8] was a main source of motivation for this work. It is one of the landmarks in producing parameterized fonts. Despite the differences between Latin and Arabic characters, we believe it is possible to apply the same concepts used in designing the CM fonts to Arabic ones.

Each character or symbol of CM has a program to describe it. The font glyphs are defined by spline vectors, but unlike current outline fonts, these vectors are defined in a clear mathematical way that can be parameterized, allowing them flexibility.

a a a a

Figure 1: Four different lowercase letter ‘a’ forms generated by a single description program. From left to right: roman, sans serif, typewriter, and bold.

The only drawback of this design technique is its difficulty. Knuth described his work to produce parameterized CM fonts to be “much, much more difficult than [he] ever imagined”. He received help from several of the world’s finest type designers, and his job, as stated by himself, was “to translate some of their wisdom into precise mathematical expressions” [8].

His final design of the CM fonts uses 62 parameters delivered to the programs describing the characters to produce 75 different standard fonts. These numbers clearly indicate the extent to which these fonts are meta-designed. Fig. 1 shows four of the lowercase letter ‘a’ generated by the CM family. These a’s and many more are output from a single description program.

We aim to produce Arabic fonts that are as meta-designed as CM. Of course parameters would be very different, for example many parameters in the CM fonts described the serifs. In Arabic there are no serifs, but instead there will be other parameters for connecting glyphs and forming ligatures.

4 Current font technologies

OpenType is currently the de facto standard font technology. It has a lot of features that support a very wide variety of languages and scripts. It is adopted by Microsoft and Adobe, and thus it is the most supported standard format. It has glyph positioning (GPOS) and glyph substitution (GSUB) tables which allow kerning and ligatures in Arabic. It also has other layout features that help in connecting glyphs in cursive scripts like Arabic. Being the most common current font standard has led to the existence of many editors and tools that help design the glyph outlines. Some tools, such as Microsoft’s Visual OpenType Layout Tool (VOLT), provide simple graphical interfaces for editing the GPOS and GSUB tables, among other features. In general, we find the main advantage of OpenType over METAFONT to be the ease of design and the availability of tools.

Despite the many features provided by OpenType, including those dedicated to the Arabic script, we see them as insufficient. The whole concept of letter boxes connecting together via other boxes of

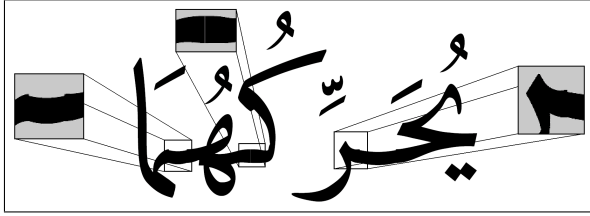


Figure 2: An example of OpenType font problems with junctions between glyphs. This word was obtained from sample products by Tradigital, a sister company of DecoType, with a typesetting system developed by Thomas Milo. Note that this font represents the highest quality in OpenType Arabic fonts we have seen, and if viewed at its standard original size, without enlargement, these imperfections are not visible.

elongation strokes is not suitable for highest quality Arabic typesetting, as we show in the following examples.

Outline fonts can be used to draw glyphs of characters in different forms very well when these glyphs are isolated. When connecting glyphs to one another, the junctions rarely fit perfectly, since adjacent letter glyphs usually have different stroke directions at the starting and ending points. Although this imperfection may not be visible for small font sizes, it is quite clear in large font sizes. An extreme example is the use of these fonts to write large banners or signs. Even for small fonts, when it is required to add a *tatweel*, a ready made *kashida* of specific length is used to connect the glyphs together. Of course, such a *kashida* will not match perfectly with all the different glyphs. Fig. 2 shows examples of problems at junctions. Two of those problems are due to using *kashidas*. The junction after the *kaf* has no *kashida*, but it shows the non-uniform stroke width. It would be possible of course to edit the outline of these two glyphs to obtain a match, but this would certainly create a mismatch with yet other glyphs.

Another limitation is the use of already stored glyphs for different ligatures; since the number of possible ligatures is very large, only a selected portion can be made available. To model the Arabic script more accurately, each word should be considered a ligature and hence we would have an almost infinite number of ligatures, which is impossible to prepare in advance. The Unicode standard has numerous glyphs called presentation forms, each representing a unique ligature form. Unicode version 5 includes around 500 codes for different glyphs, just to describe different forms of only 28 Arabic charac-

تج	سم	طم	لج	هـ	تي	نخي	كم
FD53	FD63	FD73	FD83	FD93	FDA3	FDB3	FDC3

Figure 3: Part of the character code tables indicating code allocation to complicated ligatures, combining up to 3 letters.

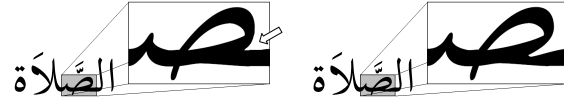


Figure 4: Optical scaling requires that stroke widths become thinner at intersections, in order to give an appearance of uniform blackness for a word at smaller scale. Left-hand figure shows a letter ‘sad’ in its medial form as it normally appears. When linearly scaled and used in a word, a black blotch appears at stroke intersections. The right-hand side shows how the ‘sad’ should be changed in order to appear properly at smaller size.

ters, not including additional codes for short vowels, diacritics, and Qur’anic marks. Fig. 3 shows some of the complex ligatures allocated codes. The provision of a code point for each ligature is an inefficient and non-scalable design in our opinion. As indicated earlier, each Arabic word can in fact be considered one ligature, so following this method of code allocation to cover all ligature forms would take up every remaining free code (and more). The process of selecting a ligature should instead be left to the typesetting application.

A final feature that is more feasible to implement in METAFONT than in OpenType is the capability to program and embed information to be used in scaling glyphs for different sizes in the fonts themselves. This additional information (called ‘hinting’ in the OpenType terminology) may be used to enable optical scaling instead of linear scaling. The optical scaling is even more important when two strokes meet, as in the medial form of the letter ‘sad’ in the left-hand side of Fig. 4. At a small scale this stroke crossing produces a black blotch when it is used in a word. Knuth [7] discussed this problem, and its solution in METAFONT by decreasing the thickness of the strokes as they intersect. This change of thickness makes the words at small sizes appear of uniform darkness; see the ‘sad’ in the right-hand side of Fig. 4. This solution can be parameterized such that, as the size decreases, the pen width at

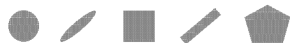


Figure 5: Various pen nib shapes in METAFONT. From left to right: circle, inclined ellipse, square, inclined rectangle, and a polygon created by the `makepen` macro.

intersections decreases, thus giving a feel of uniform darkness at all sizes.

These limitations in new font technologies led us back to METAFONT, which has existed in its current form since 1986. What new font technologies are attempting (and sometimes failing) to achieve has mostly been feasible using METAFONT since it was created. It is only due to the complexity of the task that it was not widely tried either with METAFONT or with anything else. We think that the use of the METAFONT language to produce high-quality flexible Arabic fonts might be easier than the use of current OpenType tools. In the next section we discuss one of METAFONT's most powerful features, the notion of pens.

5 Modeling pens in METAFONT

The pens used in writing Arabic are of different types and were previously discussed by Benatia *et al.* [3]. In order to solve the problems of contemporary outline fonts, we propose a better pen model to achieve an output closer to the real writing of a calligrapher. We first discuss how pen nibs are defined in METAFONT and how pen strokes are modeled.

5.1 Pen nibs in METAFONT

METAFONT provides two predefined pen nibs, for circular and square pen nib shapes: `pencircle` and `pensquare`, respectively. Each can be scaled, with different scaling factors in each direction, allowing a multitude of elliptical and rectangular shapes. The nibs can be further transformed by rotation around a specific axis. This is of extreme importance since Arabic is written using the pen nib inclined at an angle. Fig. 5 shows some of the nib shapes that can be used in METAFONT.

The most important pen macro in METAFONT is `makepen`. This macro enables the user to define any custom pen nib shape required, as long as it is a convex polygonal shape. The polygonal shape is defined by a number of coordinates connected by straight lines. The rightmost pen nib in Fig. 5 shows a polygonal nib produced by `makepen`. Since Arabic pens may not be purely rectangular or elliptical,



Figure 6: One path traced by two different pens [7]. The left-hand path was drawn using a circular pen, and the right-hand path used an elliptical pen inclined at 40 degrees from the x-axis.



Figure 7: Skeleton of the letter noon requires that the pen rotates to achieve different widths. In the left glyph (correct), the pen inclination from the x-axis changes from 70 to 120 degrees as it moves from right to left. In the right glyph (wrong), the same segment is drawn with a pen of fixed inclination of 75 degrees.

`makepen` can be used for accurate modeling of the pen nib.

After defining the pen to be used in drawing, we need to define the path to trace. In METAFONT we define points in Cartesian coordinates and then describe how the path passes through these points, in what directions and angles. Bézier curves are used by METAFONT to define the equations of these paths. Again METAFONT gives unlimited flexibility when defining paths.

5.2 Plain METAFONT drawing macros

Given the pen to be used and the path to trace, we now have the last step, the drawing action itself. The main drawing macros defined in plain METAFONT are the `draw`, `fill`, and `penstroke` macros.

John Hobby [6] developed the algorithm defining the points traced by the pen. Fig. 6 shows a path drawn using the `draw` macro but with different pens. The limitation of the `draw` macro is its use of a fixed pen inclination for the different paths in the glyph. But in Arabic calligraphy, this is not the case. Many letters require the calligrapher to change the inclination of the pen while drawing. A clear example is the skeleton of the letter noon in its extended form; see Fig. 7. Its lower segment should be thick at the middle and thin at the tips, and this requires pen rotation while tracing.

The `fill` macro simply fills a closed contour. It does not model a pen, but we will demonstrate later how it can be used within other macros to do so.

The third drawing mechanism in plain METAFONT allows rotation of the pen while tracing the path. However, this mechanism does not use the

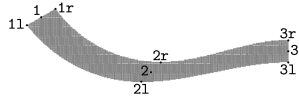


Figure 8: A path drawn using the `penstroke` macro. Note how the pen inclination changes as it moves across the path, resulting in a different path thickness at different parts.



Figure 9: The first problem with `penstroke`: razor pen

defined pen nib, and instead approximates a stroke produced by a razor pen (a pen with zero-width). This makes the underlying algorithms used to determine the shaded contour much simpler than if a polygon pen was used. This drawing mechanism uses two macros, `penpos` and `penstroke`. `penpos` defines the width and inclination of the razor pen at each coordinate pair. `penstroke` does the actual drawing depending on the `penpos` at each point; see Fig. 8. Tracing a 2-D path with a rotating polygonal pen proves to be much more complex than the case of no rotation as with the `draw` macro, and it was not implemented in the plain METAFONT macros.

Although `penstroke` solves the problem of pen rotation, the use of a razor pen leads to three other problems while drawing Arabic letters. Most notably, the zero width of the pen razor causes some unwanted effects as shown in Fig. 9 when we try to draw the letter `baa`. Close observation of the resulting glyph shows two defects directly. The first is that the left tip of the letter is too thin, indicating that the pen used has no width. The second flaw is at the bottom of the rightmost tooth of the letter intersecting with the base of the letter. This intersection is thinner than usual due to the same reason related to the pen.

Yet, this is not the only issue with `penstroke`, and not even the most prominent. There are two other problems with this macro. These problems are due to the way `penstroke` is defined in the plain METAFONT file. First, when `penpos` is used at a coordinate, METAFONT calculates the position of the left and right ends of the razor pen at each coordinate. It then forms two paths, right and left, connects them at the endpoints with straight lines, then fills the resulting contour. In fact the macro expands



Figure 10: The second problem with `penstroke`: figure-8 shape.

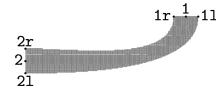


Figure 11: The third problem with `penstroke`: bad pen approximation.

to:

```
fill path_1l -- reverse path_1r -- cycle;
where path_1l is the path passing through all the
left points, and similarly for the right path.
```

This implementation causes the two problems. The first is when we try to draw a shape like that in Fig. 10. In this figure both paths intersect, resulting in the contour dividing into two (and sometimes more) regions. METAFONT does not have the capability to fill such complex regions that overlap themselves, and hence produces errors. To draw such a shape, a modification was done in our work by detecting the crossing points of the paths, then filling each region separately. Such crossings occur frequently when drawing Arabic glyphs.

The definition of `penstroke`, with both left and right paths evaluated independently, means that at some points the distance between the two paths may vary in a way that can not result from a fixed length pen. It is not always easily perceptible, but in some cases when there are sharp bends in a path or large amounts of pen rotation, the resulting stroke becomes a very bad approximation of a razor pen, as in Fig. 11. In drawing Arabic glyphs, such large rotation in pen inclination rarely occurs, but this extreme example shows that `penstroke` is not an accurate model of a razor pen. Fig. 10 also shows the same problem as a significant chunk of the stroke is missing at the middle of the path.

Although the `draw` and `penstroke` macros are good attempts to simulate pens in action, they do not fulfill the needs of Arabic pens. One does not allow pen rotation, while the other uses a pen with zero thickness. It is obvious that neither macro is sufficient, and we need the best of both worlds: a polygonal pen that traces a path while rotating. The next section discusses some proposed solutions for accurate modeling of pen strokes. These solutions

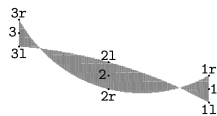


Figure 12: The error resulting from left and right paths crossing is resolved, even in the case of multiple crossings.

make use of the previously mentioned plain METAFONT macros in various ways.

5.3 Enhancements to plain METAFONT drawing macros

In this section we describe four proposed methods to provide better modeling of the pen nib tracing a path while rotating, in order of their ascending quality. With the last being the most accurate drawing method. The first method was proposed by Knuth for his CM fonts, while we developed the other three in the course of our work. But before we discuss them we will briefly mention how the errors arising from `penstrokes` left and right paths crossing, discussed in the previous section, are solved.

METAFONT does not fill a non-simple contour that crosses itself. In order to solve this, we propose finding the crossing points and then dividing the contour into segments, and filling each one separately. Since we do not know the number of crossings beforehand we do a loop until there are no more crossing points. Fig. 12 shows an example of a stroke drawn by a pen that rotates 180 degrees from point 1 to 2 and then 180 degrees more from 2 to 3, hence rotating 360 degrees in total. Such a stroke would result in an error if the plain `penstroke` is used.

5.3.1 The `filldraw stroke` macro

This technique is used a lot in Knuth’s definition of CM character glyphs. It fixes the problem of `penstroke` having zero width at certain points of a contour. Instead of just filling the `penstroke` contour, `filldraw` fills the contour and then traces its outline with a small circular nib pen, thus adding thickness to very thin segments of the “virtual pen stroke”. The reason we say it is ‘virtual’ is because Knuth’s definition of a glyph like ‘e’ keeps the pen rotating in such a way that the left and right paths of `penstroke` do not cross, and this is certainly different from what a person would do while drawing the ‘e’, hence it is not a real pen stroke.

The `stroke` macro is defined in the CM base file, and it merely defines the closed contour created by `penstroke` without filling it. Fig. 13 shows the letters e and baa’ with `penstroke` and then with `filldraw stroke`. Note the thickness effect

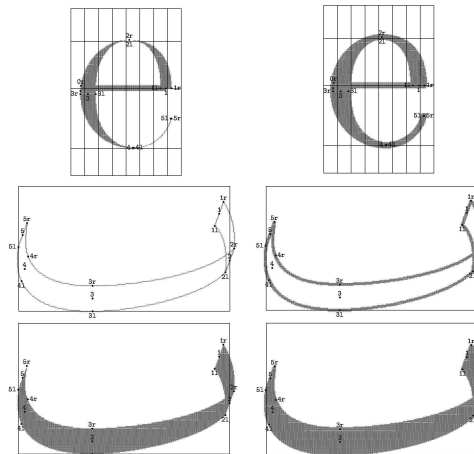


Figure 13: First solution: `filldraw`. The letter ‘e’ is shown on the top right as it is used in the CM fonts, and on the left how it would look like if drawn using `penstroke` instead of `filldraw stroke`. The top pair of baa’ letters shows the skeleton of the letter, i.e. the closed contour. The left contour is drawn with a very fine pen nib, while on the right with a thicker nib. The pair of baa’ letters at the bottom show the contours after filling.

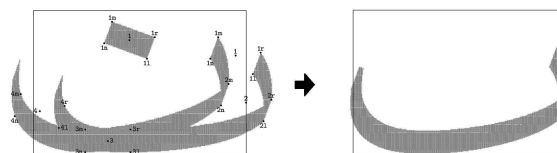


Figure 14: Second solution: `astroke`. `penpos` defines four points for each coordinate pair, named l, r, n, and m (see dot at top). `astroke` macro then produces four `penstrokes`; the left figure shows two of these sides (l-r) and (n-m).

and how Knuth used this method to give letter tips round edges. The letter baa’ is shown with its contour and after filling with `penstroke` on the left and with `filldraw stroke` on the right.

5.3.2 The `astroke` macro

Another solution to the problem of zero-width pen, is to model the pen nib with multiple `penstrokes`, one for each side of the pen. For example, for a rectangular pen nib, a macro is defined which essentially breaks into four `penstrokes`, each to model the area covered by one side of the rectangle. The `penpos` macro also had to be modified in order to evaluate the four corner points of the pen nib [l, r, n, m] instead of only two (left and right). Fig. 14 shows in the left figure two of the four `penstrokes`

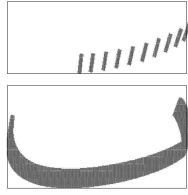


Figure 15: Third solution of stroke modeling: `qstroke`. It forms the pen stroke by drawing many instances of the pen nib along the path. The pen nib used can be any shape, not just rectangular as in the `astroke` macro.

resulting from the pen nib shown. The pen nib is enlarged for better understanding. The right figure shows all four `penstrokes` but with reasonable pen nib dimensions. The resulting baa' skeleton is much better than the one produced using only one `penstroke`, and slightly better than the one using `filldraw stroke`. Note that this macro can only model strokes for rectangular pen nibs.

A pen used to write Arabic is rarely moved in the direction of the smaller side. Hence the need to model the smaller sides of the pen is limited to tips of the glyph. This means that it is also possible to produce the same output with only two `penstrokes` (l-r) and (n-m) together with two nib dots at the start and end.

5.3.3 The `qstroke` macro

This macro solves the problem of `penstroke` being just an approximation of a razor pen traced path. The glyph is simply created by drawing footprints of the pen nib with different inclination angles at many consecutive locations along a path. The angle of the pen at each location is an interpolation of the segment's starting and ending inclination angles. At a given finite resolution, a finite number of pen dots gives the effect of a continuous pen stroke. As the distance of the path increases more pen footprints are needed. Also, since the path times in METAFONT are not linearly distributed, more instances are needed. Finally, when the pen rotation is large in a specific segment more instances are needed as well. Fig. 15 shows an example letter baa' drawn with the macro.

5.3.4 The `envelope` macro

For high resolutions, the `qstroke` macro needs to draw many dots to yield a smooth stroke. A refinement of this idea is to compute the exact envelope of the razor pen and then fill it. This `envelope` macro moves along the path at small intervals, evaluating

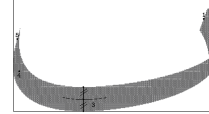


Figure 16: Fourth solution of stroke modeling: `envelope`.

at each point two equidistant corresponding points on the left and right paths, depending on the pen inclination. The output is a more accurate model of a razor pen than `penstroke`; see Fig. 16. Applying four of this new `envelope` stroke as in the `astroke` solution produces the most accurate glyph.

Of course, more accurate models require more calculations and hence more computing resources. For nominal resolutions, the `qstroke` macro will produce a final output as good as the more accurate but more complex macro, `envelope`, hence it is preferred.

Now that we have obtained a satisfactory model of the pen nib and the way it is used to draw strokes, we will explore in the next section how parameterized glyphs are designed.

6 Arabic font meta-design

With the satisfactory pen nib models of the last section, we now discuss their usage to mimic the way calligraphers draw the different letters. Our main approach is to make the writing as dynamic as possible, while obeying the traditional rules of calligraphy [2, 9]. This enables us to simulate the cursive nature of the Arabic script. In order to do that we started to design a font that is parameterized in many ways. This parameterization comes in two forms: parameterization of coordinates and of curves.

Parameterization of coordinates means that our points in the x - y plane are not given fixed locations. Any point location either depends on parameters or is related to another point in the plane sometimes also through parameters. Parameterization of curves, on the other hand, means that either the tangential direction of a curve at some points or the tension on a curve segment is dependent on parameters, or sometimes both together. This complete parameterization of the glyphs will enable us to join letters better together, extend them easily, and optically scale the font.

This process of designing the glyphs is then better described as meta-designing, since we not only design the shape of the letter, but describe how it is to be drawn, which is more difficult. Outline vector

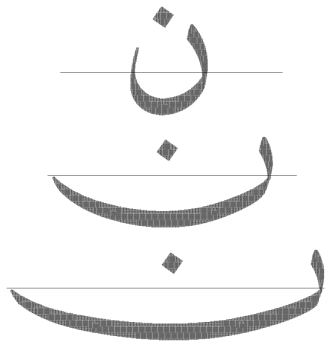


Figure 17: The letter noon shown with kasa widths of 3, 9, and 13.

fonts like TrueType can be created by merely scanning a handwritten sample and then digitizing it by converting it to vectors. The more meta-designed the glyph, the more difficult and more time it takes to describe it to the computer, but on the other hand, the better the final results become.

The Arabic alphabet, although consisting of 28 different letters, depends on only 17 different basic ‘skeletons’. Dots added above or below these skeletons differentiate one letter from another. For example the letters haa’, jeem, and khaa’ all have the same shape as haa’, but jeem has a dot below, and khaa’ has a dot above. The separation of the dots from the skeletons as well as breaking some complex skeletons to smaller parts enables us to reduce the amount of design required by considering only some primitive shapes that are repeated and used in many letters. The construction of individual letters by assembling smaller parts is the traditional method of teaching Arabic calligraphy as well.

For example, the body of the letter noon, called the kasa, is used as a part of the isolated or ending forms of many Arabic letters: seen, sheen, saad, daad, lam, and yaa’. An important property of the kasa is that it can be extended to much larger widths. Its nominal width is 3 nuqtas (Arabic dots), and when in its extended form, it can range from 9–13 nuqtas. Fig. 17 shows several instances of the longer form. Note that its width can take any real value between 9 and 13, not just integer values.

Another example of a meta-designed primitive that is used in justification is the kashida. Kashidas can be used in almost all connective letters. Here we illustrate the kashida in use with the letter haa’. Fig. 18 shows the letter haa’ in its initial form with two different kashida lengths, which differ by almost 3 nuqtas. A parameter is input to the program describing the letter in order to decide on the position

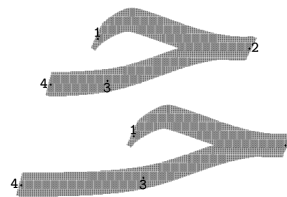


Figure 18: The initial form of the letter haa’ with two different kashida lengths.

of point 4 in relation to 3. For a longer kashida point 4 is moved to the left and also to the bottom. We will see in the next section how kashidas are set to join letters together smoothly.

7 The formation of words

After modeling the pens and meta-designing individual letters, the next logical step is to join these glyphs together to form complete words. The parameterization of the glyphs allows perfect junction points as if these glyphs were drawn with just one continuous stroke.

In the most widely used font technologies, like OpenType and TrueType, kashidas are made into ready glyphs with pre-defined lengths, and are substituted when needed between letters to give the feeling of extending the letter. But since the kashida is static, as is the rest of the surrounding letters, they rarely join well, and it is evident that the word produced is made of different segments joined by merely placing them close to one another.

In our work, the kashida is dynamic and can take continuous values, not just predefined or discrete values. Our experiments with different ways of joining various letter combinations lead us to think that when a kashida is extended between any two letters, it is neither a separate entity nor does it belong to only one of the two letters. Instead it is a connection belonging to both.

Consider for example the simple joining of the two letters, haa’ and dal (Fig. 19). Each letter is designed in a separate macro and when used to form the word, the elongation parameter of the kashida in between is passed to both macros, and the kashida is distributed on both glyphs. The two glyphs then meet at the point where the pen stroke moves exactly horizontally (parallel to the x-axis). This junction point is not necessarily at the middle of the distance between both letters. The ending point of each glyph is moved further from its letter, and in order to accommodate long kashidas, these points are moved



Figure 19: Placing a kashida between the letters haa' and dal with different lengths 2, 3, 5 and 7 dot lengths.

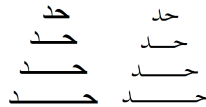


Figure 20: Placing kashidas using TrueType fonts available in Microsoft Office 2003.

slightly downwards. Long kashidas need more vertical space in order to curve smoothly, sometimes pushing the letters of a word upwards.

Other than affecting the ending points, the parameter also affects the curve definition on both sides by controlling the tensions of the paths. The resulting word at many different kashida lengths is shown in Fig. 19, which can be compared with the adding of kashidas using the TrueType fonts available in Microsoft Office 2003 as shown in Fig. 20.

Further examples of joining words can be seen in Fig. 21. This figure shows the word 'yahia' written using different fonts. Notice how the TrueType fonts connect the ligatures with a straight line, and how the OpenType font (bottom left) corrects this by placing curved kashidas. But unfortunately the curved kashidas of this OpenType font are static and do not join well. In the word produced by our parameterized font (bottom right), the letters join perfectly together, and there is also the possibility of freely extending the length between the haa' and the yaa' by any value as done in Fig. 19.

8 Future work

This paper presented new font design ideas that will enable computers to produce Arabic texts of similar quality to the works of calligraphers. The proposed parameterized font will also enable better typesetting, by providing better flexibility to the words. The work covered here is just the beginning and a small step towards the realization of such a system that produces output comparable to writings of humans writing Arabic, and much remains to be done.

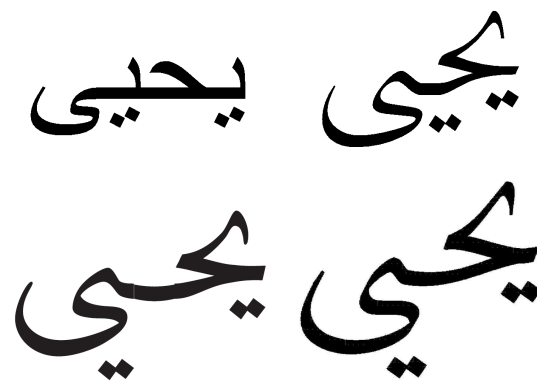


Figure 21: The word 'Yehia' as it is written using four fonts. From top left to bottom right: TrueType simplified (no ligatures), TrueType traditional, OpenType Tradigital Naskh, and our own parameterized font.

The proposed idea of producing such an output using computers opens a very vast opportunity for further research in the topic. We classify this possible future work into two categories, research within the font technology itself using METAFONT, and within the typesetting system, T_EX.

First, regarding the font design:

- Finalize meta-design of all possible letter forms. Some letters like seen and baa' have a very large variety of forms.
- Develop algorithms and method for placement of dots and diacritics, keeping in mind that their placement may, in some cases, force the calligrapher to move letters or words to free space for them. Also it should be kept in mind that this placement should not impede the legibility of the text, especially since its use is to improve legibility and understanding. Some diacritics, especially short vowels like fat-ha and kasra change their lengths and inclination, and hence are dynamic.
- Decrease the computational complexity of the current pen modeling techniques.
- Research the possibility of generating output from METAFONT other than the resolution limited bitmapped glyphs for high quality printing or screen viewing. Otherwise, to embed the METAFONT sources within new file formats such as pdf and extend the current pdf viewers to read these sources and use them to produce the correct resolution for the screen or the printer on the fly.

- Design of other writing styles besides Naskh, like Thuluth and Riq'ah, with minimal changes to the already meta-designed font.

Second, the work done in this paper together with the points mentioned above aims at the goal of providing the typesetting system with more flexibility. The typesetting engine needs some work as well:

- The selection of the most suitable glyph to be placed in a word is a very complicated task. Each letter may have many alternative forms in its specific location in the word, and these alternatives have different widths and heights. Hence, the selection of a certain form is based on many factors; most importantly, justification, and placement of dots and diacritics conflicting with ligatures. The form of the letter may be affected not only by its closest neighbors, but in some cases a letter's form may be changed depending on the fifth or sixth following letter.
- Line-breaking algorithms are a very rich topic. The flexibility in the Arabic script adds to the complexity of this task. Rules have to be added to decide whether an alternative form should be used, a ligature is to be used or broken (including which ligatures are more important than others), or where a kashida is to be added.

References

- [1] *The Holy Qur'an*. King Fahd Complex for Printing the Holy Qur'an, Madinah, KSA.
- [2] Fawzy Salem Afify. *ta'aleem al-khatt al-'arabi [Teaching Arabic calligraphy]*. Dar Ussama, Tanta, Egypt, 1998.
- [3] Mohamed Jamal Eddine Benatia, Mohamed Elyaakoubi, and Azzeddine Lazrek. Arabic text justification. *TUGboat* 27(2):137–146, November 2006. Proceedings of the 27th Annual Conference of the T_EX Users Group, Marrakesh, Morocco. <http://tug.org/TUGboat/Articles/tb27-2/tb87benatia.pdf>.
- [4] Hossam A. H. Fahmy. AlQalam for typesetting traditional Arabic texts. *TUGboat* 27(2):159–166, November 2006. Proceedings of the 27th Annual Conference of the T_EX Users Group, Marrakesh, Morocco. <http://tug.org/TUGboat/Articles/tb27-2/tb87fahmy.pdf>.
- [5] Yannis Haralambous. Simplification of the Arabic script: Three different approaches and their implementations. In *EP '98/RIDT '98: Proceedings of the 7th International Conference on Electronic Publishing, held jointly with the 4th International Conference on Raster Imaging and Digital Typography*, volume Lecture Notes 1375, pages 138–156, London, UK, 1998. Springer-Verlag. <http://omega.enstb.org/yannis/pdf/arabic-simpli98.pdf>.
- [6] John Douglas Hobby. *Digitized Brush Trajectories*. Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, CA, USA, June 1986. Also published as report STAN-CS-1070 (1985). <http://wwwlib.umi.com/dissertations/fullcit/8602484>.
- [7] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
- [8] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
- [9] Mahdy Elsayyed Mahmoud. *al-khatt al-'arabi, derasa tafseelyya mowassa'a [Arabic calligraphy, a broad detailed study]*. Maktabat al-Qur'an, Cairo, Egypt, 1995.
- [10] Thomas Milo. Arabic script and typography: A brief historical overview. In John D. Berry, editor, *Language Culture Type: International Type Design in the Age of Unicode*, pages 112–127. Graphis, November 2002. http://www.decotype.com/publications/Language_Culture_Type.pdf.

Nastaleeq: A challenge accepted by Omega

Atif Gulzar, Shafiq ur Rahman

Center for Research in Urdu Language Processing,

National University of Computer and Emerging Sciences, Lahore, Pakistan

atif dot gulzar (at) gmail dot com, shafiq dot rahman (at) nu dot edu dot pk

Abstract

Urdu is the lingua franca as well as the national language of Pakistan. It is based on Arabic script, and Nastaleeq is its default writing style. The complexity of Nastaleeq makes it one of the world's most challenging writing styles. Nastaleeq has a strong contextual dependency. It is a cursive writing style and is written diagonally from right to left. The overlapping shapes make the nuqta (dots) and kerning problem even harder.

With the advent of multilingual support in computer systems, different solutions have been proposed and implemented. But most of these are immature or platform-specific. This paper discusses the complexity of Nastaleeq and a solution that uses Omega as the typesetting engine for rendering Nastaleeq.

1 Introduction

Urdu is the lingua franca as well as the national language of Pakistan. It has more than 60 million speakers in over 20 countries [1]. Urdu writing style is derived from Arabic script. Arabic script has many writing styles including Naskh, Sulus, Riqah and Deevani, as shown in figure 1. Urdu may be written in any of these styles, however, Nastaleeq is the default writing style of Urdu. The Nastaleeq writing style was developed by Mir Ali Tabrazi in 14th century by combining Naskh and Taleeq (an old obsolete style) [2].

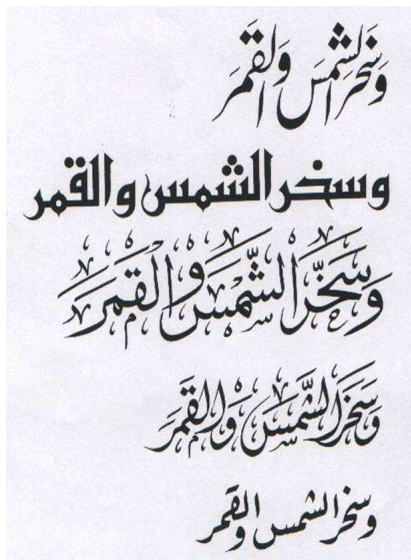


Figure 1: Different Arabic writing styles (from top to bottom: Nastaleeq, Kufi, Sulus, Deevani and Riqah) [3]

1.1 Complexity of the Nastaleeq writing style

The Nastaleeq writing style is far more complex than other writing styles of Arabic script-based languages. The salient features of Nastaleeq that make it more complex are these:

- Nastaleeq is a cursive writing style, like other Arabic styles, but it is written diagonally from right-to-left and top-to-bottom, as shown in figure 2. Numerals add to the complexity as they are written from left-to-right (figure 7).

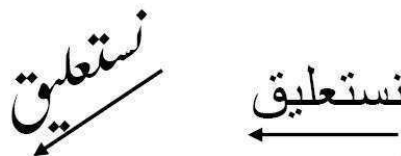


Figure 2: Direction of Nastaleeq writing style

- In most Arabic styles (especially digitized forms (fonts) of these styles), each character may assume up to four different shapes (isolated, initial, medial and final) depending on its position in the ligature. The character *Beh* (U+0628) takes four shapes depending on its position in isolated (a), initial (b), medial (c) or final (d) place in a ligature, as shown in table 1.

Nastaleeq is also a highly context sensitive writing style. The shape of a character is not only dependent on its position in a ligature but also on the shapes of the neighboring characters (mostly on the shape of the character that

ب	با	جا	جب
a	b	c	d

Table 1: Shapes of the character *Beh* at a) isolated, b) initial, c) medial, and d) final position

با	بج	بط	بی
بے	بر	بپ	بب

Table 2: Shapes of character *Beh* at initial and medial positions in different contexts

follows it). Table 2 shows a subset of the variations of *Beh* in different contexts. In Nastaleeq a single character may assume up to 50 shapes.

- In Nastaleeq some glyphs are overlapped with adjacent glyphs as shown in figure 3:

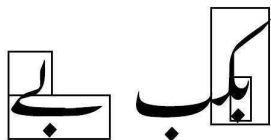


Figure 3: Overlapping glyphs in Nastaleeq

These overlapping shapes in Nastaleeq pose a major concern for kerning, proportional spacing and nuqta placement. As shown in figure 4, the ligature needs to be kerned to avoid clashing with the preceding ligature:

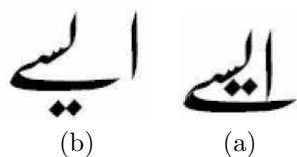


Figure 4: before (a) and after (b) kerning

- Proportional spacing is a major issue in Nastaleeq writing style. The diagonality of ligatures produces extra white space between two ligatures. Proper kerning is needed to solve that problem, as shown in figure 5.
- Nuqta placement is still another major issue in Nastaleeq rendering. Nuqtas are placed according to context, to avoid clashing with other nuqtas and boundaries of glyphs. As shown in figure 6, the nuqtas are moved downward (c) to avoid clashing with the boundary of glyph (b)) from the default position (a).



Figure 5: before (a) and after (b) kerning

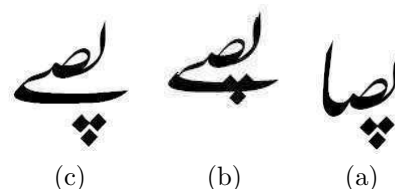


Figure 6: (a) nuqtas at default position; (b) default nuqta positioning produces a clash in different contexts; (c) default nuqtas are repositioned contextually to avoid clash.

1.2 Current Solutions

Two different techniques have been adopted for digitizing the Nastaleeq script: a ligature-based approach and a character-based approach. Each has its own limitations. The most dominant and widely used solution is the ligature-based Nori Nastaleeq. It has over 20,000 pre-composed ligatures [2]. This font can only be used with the proprietary software In-Page. The other promising solutions are character-based OpenType fonts. These fonts use OpenType technology to generate ligatures. The OpenType solution is very slow for the Nastaleeq writing style and has limitations for proportional spacing and justification.

Current solutions for the rendering of Nastaleeq script are inadequate because they do not offer consistent platform-independence and are inefficient in handling the complexity of the Nastaleeq script. These solutions are inconsistent in the sense that the results of rendering may differ from one platform to another. Currently the complete Nastaleeq solution is only available for the Windows platform. The support currently provided by Pango is quite simplistic. It implements the basic context-less initial, medial, and final rules in the OTF tables. This is no better than a Unicode font based on the Arabic presentation forms in which a character has one shape at each position. But Urdu is traditionally written in the Nastaleeq script. There is a need to provide a platform-independent solution for Nastaleeq.

The solution devised here provides Nastaleeq rendering support in Linux through Omega. Omega has the strong underlying typesetting system \TeX to handle the complexity of Nastaleeq rendering and

Omega Translation Processes (Ω TPs) provide a solution for the complexity of Nastaleeq script (e.g. contextual shape substitution) [4].

The present solution is limited to the basic alphabets of Urdu (ا (U+0627) to ل (U+06D2)) and numerals (0 to 9). These alphabets are listed in Appendix A. The solution provides:

- correct glyph substitution according to the contextual dependency of a character.
- correct cursive attachment(s) of a glyph
- nuqta placement
- automatic bidirectional support for numerals

2 Methodology

There are two possibilities for implementing support for Nastaleeq in Omega: internal Ω TPs and external Ω TPs. It is observed that internal Ω TPs are syntax dependent; for example, it is almost impossible to implement reverse chaining (processing characters/glyphs in the reverse order in a ligature) using the syntax of internal Ω TPs. External Ω TPs can be implemented using Perl or C/C++, and give the freedom to implement custom logic [4].

The solution is broadly divided into four phases. The first phase discusses the Omega virtual font generation for rendering Nastaleeq. The second and third sections discuss the contextual shape selection and smooth joining of the selected shapes. The fourth section discusses contextual nuqta placement, the most difficult feature in Nastaleeq rendering.

2.1 An Omega virtual font for Nastaleeq

An Omega virtual font file is generated from a Nafees Nastaleeq TTF font file. A total of 827 glyphs have been used to render Nastaleeq. These glyphs are placed in four different Type 1 files and four different TFM files are also generated. The Omega program itself uses only the single virtual font file `nafees.ofm` that contains pointers to the above generated font files.

2.2 Substitution logic

Nastaleeq is highly context dependent. The shape of each character in a ligature depends on the shapes of the neighboring characters. It is observed that the shape of a character is mostly dependent on the shape of the character that follows it. However, the shape of a final character in a ligature is dependent on the second to last character, with a few exceptions. For example, the character *Reh*

(ر, U+0631) has two glyphs ر and ر (as in ر)

and ر (when the character *Jeem* (ج, U+062C) occurs at the initial and medial position of a ligature, respectively. Similarly, characters U+0631, U+0691, U+0632, U+0698, U+0642, U+0648 and U+06CC all have different final glyphs depending on the glyph of the preceding character in a ligature.

In order to choose the correct glyph of a character, ligatures are processed from left-to-right, the reverse of the natural writing style of Urdu, which is right-to-left. The solution uses two lookup tables (*initial* and *medial*) to get the initial and medial shape of character according to the context. The format of these tables is shown in Table 3 below.

	U+0628	U+0629	U+0630
shape1	shape4	shape6	...
shape2	shape8	shape9	...
shape3	shape5	shape9	...
shape4	shape10	shape8	...
...

Table 3: Format of lookup table for initial and medial shape context

The first row of the table consists of Unicode values. The remainder are indices that point to the corresponding shapes in the font. For each character listed in the first row the shape of that character can be determined by looking up the shape following it, in the first column.

To find the shape of the final character two final tables are used: *final1* and *final2* for two character combinations and more than two character combinations, respectively. It is needed because final shape depends on the rightmost character; and there are only two possibilities for a character at the $(n-1)^{\text{th}}$ position: either it is an initial shape (in a two character combination) or a medial shape (in a more than two character combination).

The format of the final table is a little different from others. It has Unicode values in the first column as well, because at the beginning only Unicode values are available.

	U0628	U0629	U0630
U0628	shape4	shape6	...
U0629	shape8	shape9	...
...

Table 4: Format of lookup table for final shape context

The shape of the final character of the input string can be found by looking up the second-to-last character of the input string in the first column.

The first step for substitution is to break the input string into ligature strings. Ligatures are then processed from left to right as follows:

For a ligature of length n , the shape of the n^{th} character is recognized by consulting the final tables.

```
//if there are more than two characters
if (n>1)
  ligature[n] = final2[lig[n]][lig[n-1]]
//if there are only two characters
elseif (n>0)
  ligature[n] = final[lig[n]][lig[n-1]]
```

Where the *lig* string consists of Unicode values of characters in a ligature and the *ligature* string holds the shapes of these characters.

For the remaining $n - 2$ characters, the medial table is consulted. The shape of the n^{th} character can be found in the medial table as follows:

```
for (k=n-1; k>0; k--) {
  ligature[k] =
  medial[mrcompress[ligature[k+1]][lig[k]]
}
```

Where *mrcompress* is the compressed medial table.

The shape of first character in a ligature can be found by consulting the initial table:

```
ligature[0] =
  initial[ircompress[ligature[1]][lig[0]]
```

where *ircompress* is the compressed initial table.

Finally the ligature is checked to see if it is composed of numerals. In case of numerals, the string is printed in reverse order, so as to maintain the direction of numeric characters — from left to right.

```
if (ligature is composed of
  numeric characters)
  for (i=n; i>=0; i--)
    Output ligature[i]
```

پاکستان ۴ اگست ۱۹۴۷ء کو معرض وجود میں آیا

Figure 7: Sample string with numeric characters

2.3 Positioning

Nastaleeq is written diagonally from right-to-left and top-to-bottom. The baseline of Nastaleeq writing style is not a straight horizontal line; instead, the baseline of each glyph is dependent on the baseline of following glyph. Similarly, the position of a particular glyph is relative to the position of the glyph following it.

TeX does not know anything about the shape of a character. It only knows the box with height, width and depth properties. TeX output contains a list of boxes concatenated with each other. By default these boxes are aligned along the baseline (Fig. 8). But these boxes can be shifted horizontally or vertically.

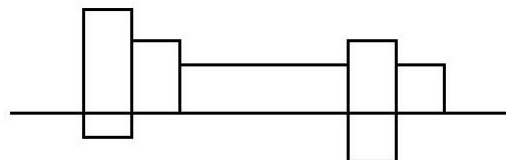


Figure 8: TeX boxes

The devised solution uses the pre-computed entry and exit points of glyphs that are stored in a file. Entry points are points where the immediate right-hand glyph should connect; similarly, exit points represent the points where the immediate left-hand glyph should connect.

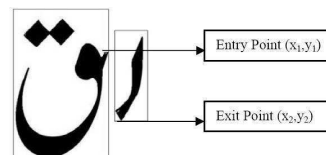


Figure 9: Entry and exit points

In the above example the vertical adjustment for the right-hand glyph will be $y_1 - y_2$. And the resulting output is shown in figure 10:

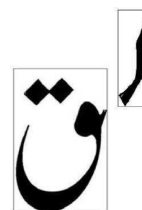


Figure 10: After vertical adjustment

Similarly, the horizontal adjustment can also be made for proper cursive attachment between two consecutive glyphs:



Figure 11: After vertical and horizontal adjustment

Two passes are needed for proper glyph positioning in a ligature. For vertical positioning the ligature is processed from left-to-right. It is done this way because the n^{th} (last) glyph of a ligature always resides on the baseline, while the other $n - 1$ glyphs move vertically upward according to the entry and exit points.

```
y=0;
for (j=n; j>=1; j--) {
  y = enex[ligature[j]][1]
    - enex[ligature[j-1]][3] + y;
  ligenex[j-1][1] = y;
}
```

where the *enex* table contains the entry and exit points, the *ligenex* table holds the resultant cursive attachments and *ligature* contains the shape indices of ligature.

In the 2nd pass the ligature is processed from right-to-left for horizontal positioning. The first glyph of a ligature is positioned horizontally with respect to the previous ligature and then the remaining $n - 1$ glyphs are kerned for smooth joining.

```
for (j=0; j<n; j++) {
  ligenex[j+1][0] =
    (enex[ligature[j]][2]
    +enex[ligature[j+1]][0])
}
```

Kerning is another major issue in Nastaleeq rendering. There are two kinds of kerning problems: one produces extra space between ligatures (a), and the other creates a clash between ligatures (b). Case (a) is not included in the present implementation, but case (b) is handled.

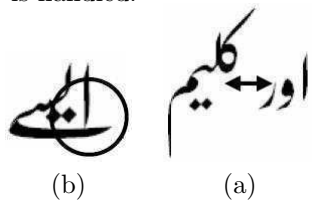


Figure 12: Types of kerning problems

The final shapes of the characters *Yeh Barree* (ے, U+06D2), *Jeem* (ج, U+062C) and *Ain* (ع, U+0639) produce (in some cases) negative kerning, which results in clashes with the preceding ligature. To avoid such clashes a positive kerning is made. The factor of this kerning is calculated by subtracting the width of final glyph from the sum of widths of the preceding $n - 1$ glyphs of the same ligature, as shown in figure 4.

$kern = width[n-1] - width \text{ of final glyphs}$

where *kern* is the positive kerning value for a ligature of length n , where *width*[x] holds the aggregate widths of x glyphs.

2.4 Contextual nuqta placement

Nuqta placement is the most complex problem of Nastaleeq rendering. Due to overlapping shapes, nuqtas cannot be placed at fixed positions, but must be adjusted according to the context. Thus, nuqtas are stored separately from the base glyph. There are two major kinds of nuqta problems: nuqta collision with the neighboring glyph (a) and nuqta collision with adjacent nuqtas (b), as shown in figure 13:



Figure 13: Nuqta collision types

Initially nuqtas are placed at the most natural position (figure 14) for individual glyphs. Nuqtas are then adjusted for the above two problems.



Figure 14: Nuqta placement at default positions

There are 26 characters in Urdu that have nuqtas, as shown below; character *Yeh* (ے, U+064A) has nuqtas at only its initial and medial position.

ب، پ، ت، ٹ، ث، ج، چ، خ، ڈ، ذ، ژ، ز، ش،
ظ، ض، غ، ف، ق، ن، ی

The intra-ligature clashes of nuqtas with the neighboring characters are handled case by case. Our investigations found that the following characters influenced the nuqta positioning due to the shape of their glyphs.

ے، ج، چ، ح، خ، ف، ق، ع and ک

For example, the final glyph of *Yeh Barree* (ے) produced problems for the nuqta characters that are vertically overlapped over the shape of *Yeh Barree*. To avoid this problem all such nuqtas are placed below the horizontal strike of the *Yeh Barree* shape, as shown in figure 15.

Nuqta clashes are removed according to following observations.

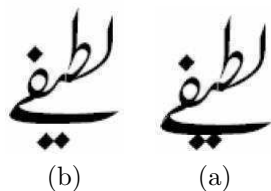


Figure 15: Nuqta placement for *Yeh Barree*

- The nuqtas of final letters are usually not displaced.
- The nuqtas of isolated letters are usually not displaced.
- The nuqtas of *dad* (U+0639) and *zah* (U+0638) are not displaced.
- Nuqtas of initial letters are preferably placed in their position.
- Nuqtas' clashes with neighboring characters are handled case by case.
- The nuqtas are displaced right (preferably) in case of clash with neighboring nuqtas.
- If the displaced nuqtas are confused with the next letter or clashes, the nuqtas are moved downwards (or upwards) instead of horizontally.

3 Results and discussions

There are more than 20,000 valid ligatures in Urdu. The sample data of approximately 7,000 ligatures is randomly selected from the corpus of 20,000 valid ligatures. The data is tested for correct contextual substitution, cursive attachment and nuqta placement. The next table shows the test results for the following test points.

- Proper glyph is substituted
- There is a smooth cursive join between glyphs
- Nuqtas are positioned at the right place without clashing with another nuqta or the boundary of a glyph.

The test results are shown in table 5.

4 Future enhancements

This work will provide a platform for the following future enhancements.

- Support for diacritics
- Proportional spacing across ligatures
- Justification
- Improvements in nuqta placement

Number of characters in a ligature	Number of ligatures tested	Incorrect substitution	Incorrect positioning	Nuqta clash
8	26	0	0	1
7	253	0	0	5
6	1545	0	0	20
5	1500	0	0	18
4	1500	0	0	15
3	1500	0	0	5
2	600	0	0	0
total	7000	0	0	65

Table 5: Test results

Acknowledgement

We would like to thank the Nafees Nastaleeq font development team, especially the calligrapher Mr. Jamil-ur-Rehman who created the beautiful glyphs for this font. The beauty of this font gave us the inspiration to provide Nafees Nastaleeq rendering support in Linux through Omega.

References

- [1] <http://www.ethnologue.com>
- [2] <http://en.wikipedia.org/wiki/Nastaliq>
- [3] *Urdu calligraphy and fonts* by Sarmad Hussain at Urdu Fonts Development Workshop, 2003. <http://www.tremu.gov.pk/tremu/workinggroups/presentation.htm>
- [4] *Draft Document for the Ω system*, by John Plaice, Yannis Haralambous, March 1999.

Appendix A

Characters in scope are listed in the table below.

U+0622	U+0627	U+0628	U+067E	U+062A
U+0679	U+062B	U+062C	U+0686	U+062D
U+062E	U+062F	U+0688	U+0630	U+0631
U+0691	U+0632	U+0698	U+0633	U+0634
U+0635	U+0636	U+0637	U+0638	U+0639
U+063A	U+0641	U+0642	U+06A9	U+06AF
U+0644	U+0645	U+0646	U+06BA	U+0648
U+06C1	U+06BE	U+0626	U+06CC	U+06D2
U+06F0	U+06F1	U+06F2	U+06F3	U+06F4
U+06F5	U+06F6	U+06F7	U+06F8	U+06F9

Typesetting Vietnamese with VnTeX (and with the TeX Gyre fonts too)

Hàn Thế Thành
River Valley Technologies
<http://river-valley.com>

Abstract

This paper attempts to give an overview of the topics related to typesetting Vietnamese with TeX. There are no detailed instructions, but rather a comprehensive list of relevant issues and resources, so the reader can get an overall feeling of what is possible and where to look for further information for a quick start. A section is dedicated to a frequently asked question, i. e. how to write a few Vietnamese words in a non-Vietnamese document. I also take a close look at the Latin Modern and TeX Gyre fonts from the perspective of a Vietnamese end user, hoping to provide some useful feedback to the authors of those fonts.

1 A brief introduction to the Vietnamese language

For convenience I repeat here a short introduction to the Vietnamese language from [1] (with small modifications):

Vietnamese is written with Latin letters and additional accents in a system called *Quốc Ngữ* (which can be translated to English as “national language”) developed by the French missionary Alexander de Rhodes. What separates Vietnamese from other languages typeset with Latin characters is that in Vietnamese some letters can have two accents. The total number of accented letters in Vietnamese (including uppercase and lowercase letters) is 134. Table 1 lists all accented lowercase Vietnamese letters.

Vietnamese accents can be divided into three kinds of diacritic marks: tone, vowel and consonant. Table 2 shows them all with examples.

The accents in Vietnamese play a very important role, since they are used to indicate different meanings and pronunciations of words. The same words with different accents can have a totally different meaning (though the pronunciations can sound very similar to a non-Vietnamese).

The vowel marks are more important than tone marks. Letters with different vowel marks are considered as distinguished letters, but a letter with different tone marks is considered as a single letter with different tones. Vowels have more impact on pronunciation than tones.

The official Vietnamese alphabet as taught in school has 29 letters in this collating order:

A Ă Â B C D Đ E Ê G H I K L M N
O Ô Ó P Q R S T U Ú V X Y
a ă â b c d đ e ê g h i k l m n
o ô ó p q r s t u ú v x y

In Vietnamese all words consist of single syllables, so they are often very short; hyphenation is not allowed at all.

The large number of Vietnamese accented letters has caused big problems in the past. Since there are 134 accented letters, we have 6 letters that don’t fit within the upper 128 positions of an 8-bit encoding. So people invented various encodings to deal with this problem, using techniques like dropping uppercase letters, or placing some letters in slots 0–31, or removing some ASCII letters. None was perfect, until UTF-8 became widely used. Thanks to UTF-8 support in the recent L^ATeX kernel, this is no longer a problem.

The way that punctuation marks are typeset is not consistent; sometimes people put a space before them, sometimes not. However, the case without a space before punctuation is dominant.

More information on the Vietnamese alphabet can be found in [9].

2 Introduction to VnTeX

VnTeX is a package to typeset Vietnamese with L^ATeX and plain TeX. VnTeX consists of the following:

- Vietnamese fonts;
- L^ATeX support for Vietnamese: input encoding, font encoding, support for babel, a style file to activate Vietnamese, etc.;
- plain TeX support;
- comprehensive font samples and test files;
- some brief documentation.

a	á	ạ	à	ả	ã
ă	ắ	ặ	ằ	ẳ	ẵ
â	ấ	ậ	ầ	ẩ	ẫ
e	é	ẹ	è	ẻ	ẽ
ê	ế	ệ	ề	ể	ễ
i	í	ị	ì	ỉ	ĩ
o	ó	ọ	ò	ỏ	õ
ô	ố	ộ	ồ	ỗ	ỗ
ơ	ớ	ợ	ờ	ở	ỡ
u	ú	ụ	ù	ủ	ũ
ư	ứ	ự	ừ	ử	ữ
y	ý	ỵ	ỳ	ỷ	ỹ
đ					

Table 1: List of all Vietnamese lowercase letters taking accents

Vowel	
breve	lăn
circumflex	tôi
horn	lươn
Tone	
acute	lá
grave	là
hook above	lả
tilde	lã
dot below	lạ
Consonant	
stroke	đi

Table 2: List of all Vietnamese diacritic marks

VnTeX tries to make typesetting Vietnamese as accessible and easy as typesetting English, and also tries to follow common L^AT_EX conventions, in order to minimize the chance of conflicts with other packages. The official L^AT_EX font encoding for Vietnamese is T5, made by Werner Lemberg and Vladimir Volovich. VnTeX supports various input encodings for Vietnamese. The recommended encoding to use with VnTeX is UTF-8. If for some reason UTF-8 cannot be used (for example your T_EX editor doesn't support it), VISCI is the recommended 8-bit encoding.

The first version was released in 2000. The current version is 3.1.5 and was released in January 2007. VnTeX is being maintained by Hàn Thê Thành, Werner Lemberg and Reinhard Kotucha. The official website of VnTeX is [2].

VnTeX is already part of T_EX Live and MiK_TE_X, which frees most users from the need to install it manually.

ConT_EXt also has support for Vietnamese (partly imported from VnTeX) and a number of Vietnamese users.

3 How to typeset just a few Vietnamese words

This section tries to answer the question that has often been asked: *How can I typeset just a few Vietnamese words in my L^AT_EX document, which is written in English (German/Polish/French/...)?*

The answer depends very much on the particular scenario; however, I assume that you are in a hurry, you don't want to bother with issues like how to display and write Vietnamese in your T_EX editor. You have only a few Vietnamese words in your L^AT_EX file and you would like to see them properly displayed

in your final PDF or PostScript file.

- As the very first requirement, you must have some minimal L^AT_EX support for Vietnamese:
 - Check whether you have VnTeX installed. VnTeX is included in t_EX, MiK_TE_X and T_EX Live.
 - If you don't have VnTeX installed and you are using Latin Modern or T_EX Gyre fonts, then you can just download a single file <http://vntex.sf.net/download/vntex-support/t5enc.def> and put it in the directory where your L^AT_EX source is.
 - Or, if you feel that you need VnTeX, or want to give it a try, or you are not using Latin Modern or T_EX Gyre fonts but a font available in VnTeX only, you can try to download and install VnTeX by following the instructions (in English) at [3].
- If all the above fails, try to get help from someone else to solve at least one of those issues.
- Make sure you have the package `fontenc` loaded with T5 encoding. For example, if your document contains European language(s) only, then you should have a line saying


```
\usepackage[T5,T1]{fontenc}
```

 in your preamble.
- Here's an example of how to input Vietnamese words:

```
{\fontencoding{T5}\selectfont
Ti'\ecircumflex{ng Vi\d\ecircumflex}{t}
which gives the output Tiếng Việt.
```

- If the font family you are using doesn't support T5 encoding, you might want to use another family by saying something like

```
\fontencoding{T5}\fontfamily{cmr}
\selectfont
```

Instead of `cmr` (Computer Modern fonts) you can use `lmr` (Latin Modern fonts), or any font family that supports T5 encoding. For a complete list, see [4].

- Table 3 contains all Vietnamese letters for your reference.
- If you have quite a lot of Vietnamese words it can be somewhat tedious to translate them to the above form (often called the L^AT_EX Internal Character Representation — LICR). On Windows you can use the package <http://vntex.sourceforge.net/download/vntex-support/tovntex.zip> to translate text in the clipboard from VIQR or UTF-8 to LICR by one key press.

The same (or close) convenience could be made for Unix/Linux users, but at somewhat higher cost due to deficiencies of Unix-like systems. So if you don't use Windows then you are out of luck, sorry. However, if you use Vim, you can still download the package mentioned above, and use the `vim` script inside the zip archive to do the conversion. If you want to make this easier for Unix users then let me know.

In the future, a web page might be set up to provide online conversion from UTF-8 text to LICR.

4 A quick start for VnTeX users

This section describes the very first steps for those who want to use VnTeX. In contrast to the previous section, the requirement here is that you have VnTeX installed, your editor can properly display Vietnamese text and you can input Vietnamese text. If your setup doesn't meet these requirements, here are some quick tips:

- for Windows users:
 - install MiKTeX;
 - use a text editor that supports UTF-8: TeX-Maker with a Unicode font like Courier New;
 - use the Unikey keyboard driver to input Vietnamese letters.
- for Unix users:
 - install T_EX Live 2007;
 - use a text editor that supports UTF-8: TeX-Maker, Emacs or Kile, with a Unicode font like Courier New;

- use the XUniKey or Xvnc keyboard driver to input Vietnamese letters (this is not needed for Emacs which has its own input methods).

Supposing that you have VnTeX installed and can read/write Vietnamese with your editor, we can continue now.

A minimal example looks as follows:

```
\documentclass{report}
\usepackage[utf8]{vietnam}
\begin{document}
Tiếng Việt
\end{document}
```

If the document contains multiple languages, you can use `babel`:

```
\documentclass{report}
\usepackage[english,vietnam]{babel}
\usepackage[utf8]{inputenc}
\begin{document}
\selectlanguage{english}
English text
\selectlanguage{vietnam}
Tiếng Việt
\end{document}
```

Once you get started with those examples, you might be interested in some practical issues like:

- how to use VnTeX with Scientific WorkPlace,
- how to create PDF that can be searched, cut or pasted,
- how to create PDF with Unicode bookmarks,
- how to use MakeIndex with VnTeX,
- how to convert L^AT_EX to HTML using T_EX4ht.

All those issues are solved and described at the VnTeX website [2].

5 A survey of fonts that can be used with VnTeX

VnTeX comes with quite a large set of fonts. Apart from that, many existing fonts also support Vietnamese. Most of them are made by the Polish font experts who are working on the Latin Modern and T_EX Gyre font projects.

5.1 Fonts coming with VnTeX

Many freely available T_EX fonts have their Vietnamese counterpart coming with VnTeX:

- VNR: based on Computer Modern, the default T_EX fonts;
- URWVN: based on URW fonts, the free version of 35 standard PostScript fonts;
- Vn CM Bright: based on CM Bright, a sans serif font derived from Computer Modern fonts;

À \‘A	Í \‘I	Ý \‘Y	ì \d{i}
Ả \h{A}	Ị \d{I}	Ỡ \d{Y}	ò \‘o
Ã \~A	Ỡ \‘o	à \‘a	Ỡ \h{o}
Á \‘A	Ỡ \h{0}	ả \h{a}	Ỡ \~o
Ạ \d{A}	Ỡ \~o	ã \~a	ó \‘o
À \Abreve	Ỡ \‘o	á \‘a	ơ \d{o}
Ả \‘\Abreve	Ỡ \d{0}	ạ \d{a}	Ỡ \ocircumflex
Ã \h\Abreve	Ỡ \Ocircumflex	ă \abreve	Ỡ \‘\ocircumflex
Ã \~\Abreve	Ỡ \‘\Ocircumflex	ã \‘\abreve	Ỡ \h\ocircumflex
Á \‘\Abreve	Ỡ \h\Ocircumflex	ã \h\abreve	Ỡ \~\ocircumflex
Ạ \d\Abreve	Ỡ \~\Ocircumflex	ã \~\abreve	Ỡ \‘\ocircumflex
Ả \Acircumflex	Ỡ \‘\Ocircumflex	ă \d\abreve	Ỡ \d\ocircumflex
Ả \‘\Acircumflex	Ỡ \d\Ocircumflex	â \acircumflex	ơ \ohorn
Ả \h\Acircumflex	Ỡ \Ohorn	â \‘\acircumflex	Ỡ \‘\ohorn
Ã \~\Acircumflex	Ỡ \‘\Ohorn	â \h\acircumflex	Ỡ \h\ohorn
Á \‘\Acircumflex	Ỡ \h\Ohorn	â \h\acircumflex	Ỡ \~\ohorn
Ả \d\Acircumflex	Ỡ \~\Ohorn	ã \~\acircumflex	Ỡ \‘\ohorn
Đ \DJ	Ỡ \‘\Ohorn	â \d\acircumflex	ơ \d\ohorn
È \‘E	Ỡ \d\Ohorn	đ \dj	ù \‘u
Ê \h{E}	Û \‘U	è \‘e	ủ \h{u}
Ê \~E	Û \h{U}	ê \h{e}	ũ \~u
É \‘E	Û \~U	ẽ \~e	ú \‘u
Ẹ \d{E}	Ú \‘U	é \‘e	ụ \d{u}
Ê \Ecircumflex	Ụ \d{U}	ẹ \d{e}	ư \uhorn
Ê \‘\Ecircumflex	Û \Uhorn	ê \ecircumflex	ừ \‘\uhorn
Ê \h\Ecircumflex	Û \‘\Uhorn	è \‘\ecircumflex	ử \h\uhorn
Ê \~\Ecircumflex	Û \h\Uhorn	ê \h\ecircumflex	ừ \~\uhorn
É \‘\Ecircumflex	Û \h\Uhorn	ẽ \~\ecircumflex	ứ \‘\uhorn
Ẹ \d\Ecircumflex	Û \~\Uhorn	ê \~\ecircumflex	ư \d\uhorn
Ì \‘I	Û \‘\Uhorn	ê \d\ecircumflex	ỳ \‘y
Ỉ \h{I}	Ỡ \d\Uhorn	ì \‘i	ỷ \h{y}
Ĩ \~I	Ỡ \‘Y	ỉ \h{i}	ỷ \~y
	Ỡ \h{Y}	ĩ \~i	ỷ \d{y}
	Ỡ \~Y	í \‘i	

Table 3: Vietnamese alphabet

- Vn Concrete: based on the CM Concrete font;
- TXTTVN: based on TXTT, a very nice type-writer font from the `txfonts` package;
- Vntopia: based on Adobe Utopia;
- Vn Charter: based on Bitstream Charter;
- Vn URW Grotesk: based on URW Grotesk font, a heavy font suitable for displayed text;
- Vn Garamond: based on URW Garamond;

- Vn Classico: based on URW Classico (also known as Optima).

In addition to the above, VnTeX also contains T_EX support for some TrueType fonts (MS Core fonts, ArevSan, ComicSans).

5.2 Fonts from other sources

The fonts made by Bogusław Jackowski and Janusz

Nowacki have excellent support for many languages including Vietnamese. The fonts come ready-to-use and cover all Vietnamese letters. Since the fonts are so well known, I will only list them here without description:

- Latin Modern fonts,
- T_EX Gyre fonts (Bonum, Pagella, Schola, Termes, Heros),
- Antykwa Toruńska,
- Iwona,
- Kurier.

Antykwa Półtawskiego is also a very nice font made by our Polish font experts but doesn't support Vietnamese yet. Therefore I take this opportunity to repeat my plea to add Vietnamese support to this font :).

Starting with version 7.0, Adobe Reader comes with two nice fonts: Minion and Myriad. These fonts have full support for Vietnamese. However, the use of these fonts is restricted to Adobe Reader only. It's not even allowed to embed a subset of the fonts in a PDF or PostScript file. Using a recent version of pdfT_EX, it is possible to use the fonts without embedding them (but doing so implies that the PDF output must be viewed or printed with Adobe Reader 7.0 or newer).

5.3 Typesetting Vietnamese and math

As mentioned above, VnT_EX tries to make typesetting Vietnamese as easy as typesetting English. When it comes to math typesetting, this holds true as well. Most of the fonts listed in the Free Math Font Survey [7] either have a Vietnamese version available with VnT_EX, or support Vietnamese themselves. Therefore, the samples listed in the survey also apply to Vietnamese. There is even a Vietnamese translation of the survey [8].

6 Comments on Latin Modern and T_EX Gyre from a user perspective

The Latin Modern and T_EX Gyre fonts try to make life easier for T_EX maintainers as well as T_EX users. Therefore I think it is important for the authors to get feedback not only from font experts and T_EX maintainers, but also from end users. In this section I would like to comment on the fonts from the perspective of a Vietnamese T_EX user.

Ease of use: excellent, since the fonts are already included in major T_EX distributions, with everything needed for use. In other words, it cannot be easier for someone who wants to use them or give them a try. However, it seems that people

are not very interested in trying, since the benefits of switching to those new fonts are not that clear for an end user. Perhaps a short document explaining the benefits to switch to those new fonts would be helpful.

Quality: the Vietnamese letters look very good in general:

- accents are very consistent regarding shape and placement;
- in most cases the accents look similar to or better than those in VNR and URWVN fonts;
- kerning data are added for Vietnamese letters as well;
- the widths of letters like Ū, Ō, ū, σ are adjusted properly;
- the quality of hints and outlines are far better than in URWVN fonts.

Issues: still, there are some issues regarding accent shapes and placement that need to be discussed and perhaps re-considered if applicable:

- sometimes the accents don't seem to have the same (or optically compatible) darkness; especially the grave accent is often darker than the others in italic fonts;
- the placement of acute and grave over circumflex does not seem optimal to Vietnamese users, especially the combination of grave over circumflex. It is preferable to have the grave rather on the right side of the circumflex, instead of the center.
- sometimes the accents are placed too close to the base letter or to each other, making some letters look too "crowded". This is more visible at small sizes, or low resolution.

7 Acknowledgments

I would like to thank all the people who have contributed to VnT_EX in various ways. I try to list here the most significant contributors that I can recall, in no particular order:

- Vladimir Volovich for work on the T5 encoding and supporting T5 encoding in his `cmmap` package,
- Werner Lemberg for major L^AT_EX support in VnT_EX and also for the `vncmr` package, from which VnT_EX was derived,
- Reinhard Kotucha for maintaining the VnT_EX package,
- Bogusław Jackowski and Janusz M. Nowacki for their excellent support of Vietnamese in their fonts,

- Nguyễn Đại Quý (also known as `vnppenguin`) for supporting the `vntex.org` domain and many various issues related to Vn \TeX , including dedicating a box on his forum [5] to Vn \TeX ,
- Thái Phú Khánh Hòa for the Vietnamese translation of the Free Math Font Survey and the Vn \TeX logo at the Vn \TeX website [2].

References

- [1] Hàn Thế Thành, *Making Type 1 fonts for Vietnamese*, *TUGboat* 24:1, Proc. of the 24th Annual Meeting and Conference of the \TeX Users Group, pp. 69–84.
- [2] The official Vn \TeX website: `vntex.sf.net`.
- [3] The latest version of Vn \TeX and installation instructions (in English) are available at: `http://vntex.sf.net/download/vntex`
- [4] The complete font samples that can be used with Vn \TeX are available at `http://vntex.sf.net/fonts/samples`. The NFSS and TFM names of all fonts are included, making it easy to use a particular font.
- [5] The VnOSS forum has a box dedicated to \TeX and Vietnamese:
`http://forum.vnoss.org/viewforum.php?id=10`
- [6] Another web site with a forum useful for Vietnamese \TeX users:
`http://vietttug.org`
- [7] The Free Math Font Survey is available at:
`http://ctan.tug.org/tex-archive/info/Free_Math_Font_Survey/survey.html`
- [8] The Vietnamese translation of the Free Math Font Survey is available at:
`http://ctan.org/tex-archive/info/Free_Math_Font_Survey/vn/survey-vn.pdf`
- [9] The definition of the Vietnamese alphabet at Wikipedia: `http://en.wikipedia.org/wiki/Quoc_ngu`

Managing order relations in MIBIB \TeX *

Jean-Michel Hufflen

LIFC (EA CNRS 4157)

University of Franche-Comté

16, route de Gray

25030 BESANÇON CEDEX

France

`hufflen (at) lifc dot univ-fcomte dot fr`

`http://lifc.univ-fcomte.fr/~hufflen`

Abstract

Lexicographical order relations used within dictionaries are language-dependent. First, we describe the problems inherent in automatic generation of multilingual bibliographies. Second, we explain how these problems are handled within MIBIB \TeX . To add or update an order relation for a particular natural language, we have to program in Scheme, but we show that MIBIB \TeX 's environment eases this task as far as possible.

Keywords Lexicographical order relations, dictionaries, bibliographies, collation algorithm, Unicode, MIBIB \TeX , Scheme.

Streszczenie

Porządek leksykograficzny w słownikach jest zależny od języka. Najpierw omówimy problemy powstające przy automatycznym generowaniu bibliografii wielojęzycznych. Następnie wyjaśnimy, jak są one traktowane w MIBIB \TeX -u. Dodanie lub zaktualizowanie zasad sortowania dla konkretnego języka naturalnego umożliwia program napisany w języku Scheme. Pokażemy, jak bardzo otoczenie MIBIB \TeX -owe ułatwia to zadanie.

Słowa kluczowe Zasady sortowania leksykograficznego, słowniki, bibliografie, algorytmy sortowania leksykograficznego, Unikod, MIBIB \TeX , Scheme.

0 Introduction

Looking for a word in a dictionary or for a name in a phone book is a common task. We get used to the lexicographic order over a long time. More precisely, we get used to *our own* lexicographic order, because it belongs to our cultural background. It depends on languages. This problem is particularly acute when we deal with managing multilingual bibliographies, as in our program MIBIB \TeX — for ‘MultiLingual BIB \TeX ’. Let us recall that this program aims to be a ‘better’ BIB \TeX [15], the bibliography processor usually associated with the L \TeX word processor [12]. When it builds a ‘References’ section for a L \TeX document, BIB \TeX uses a bibliography style to determine the layout. Some bibliography styles are *unsorted*, that is, the order of bibliographical items within the bibliography is the order of first citations of these items throughout the document. However, most of BIB \TeX 's styles sort

these items w.r.t. the alphabetical order of authors' names. But the `bst` language of bibliography styles [14] only provides a `SORT` function [13, Table 13.7] suitable for the English language, the commands for accents and other diacritical signs being ignored by this sort operation.

The purpose of this article is to show how this problem is solved in MIBIB \TeX 's first public release. In practice, this version deals only with European languages using the Latin alphabet. The MIBIB \TeX program is written using the Scheme programming language [10]. A standardised library providing support for Unicode [22] has been designed [18, §§ 1.1 & 1.2], but we cannot say that the present version of Scheme is Unicode-compliant, even if some implementations include partial support.¹ So some parts of our present implementation of order relations are temporary, but we think that this implementation

* Title in Polish: *Zarządzanie zasadami sortowania leksykograficznego w MIBIB \TeX -u.*

¹ At the time of finishing the revised version of this article, the proposal for Scheme's next standard has just been ratified and is now the ‘official’ sixth version of this language [19, 18]. See <http://www.r6rs.org> for more details.

- The Czech alphabet is: $a < b < c < \check{c} < d < \dots < h < ch < i < \dots < r < \check{r} < s < \check{s} < t < \dots < z < \check{z}$.
 - In Danish, accented letters are grouped at the end of the alphabet: $a < \dots < z < \ae < \emptyset < a$.
 - The Estonian language does not use the same order for unaccented letters as the usual Latin order; in addition, accented letters are either inserted into the alphabet or alphabeticised like the corresponding unaccented letter: $a < \dots < s \sim \check{s} < z \sim \check{z} < t < \dots < w < \bar{o} < \check{a} < \bar{o} < \bar{u} < x < y$.
 - Here are the accented letters in the French language: $\grave{a} \sim \hat{a}, \grave{c}, \grave{e} \sim \acute{e} \sim \hat{e} \sim \ddot{e}, \hat{i} \sim \bar{i}, \hat{o}, \grave{u} \sim \hat{u} \sim \ddot{u}, \ddot{y}$.
When two words differ by an accent, the unaccented letter takes precedence, but w.r.t. a right-to-left order:^a $cote < c\acute{o}te < cot\acute{e} < c\acute{o}t\acute{e}$. Two ligatures are used: ‘æ’ (resp. ‘œ’), alphabeticised like ‘ae’ (resp. ‘oe’).
 - There are three accented letters in German — ‘ä’, ‘ö’, ‘ü’ — and three lexicographic orders:
 - DIN^b-1: $a \sim \check{a}, o \sim \check{o}, u \sim \check{u}$;
 - DIN-2: $ae \sim \check{a}, oe \sim \check{o}, ue \sim \check{u}$;
 - Austrian: $a < \check{a} < \dots < o < \check{o} < \dots < u < \check{u} < v < \dots < z$.
 - The Hungarian alphabet is:

$$a \sim \acute{a} < b < c < cs < d < dz < dzs < e \sim \acute{e} < f < g < gy < h < i \sim \acute{i} < j < k < l < ly < m < n < ny < o \sim \acute{o} < \check{o} \sim \check{o} < p < \dots < s < sz < t < ty < u \sim \acute{u} < \check{u} \sim \check{u} < v < \dots < z < zs$$
- Some double digraphs must be restored before sorting:
- $$ccs \rightarrow cs+cs, ddz \rightarrow dz+dz, ggy \rightarrow gy+gy, lly \rightarrow ly+ly, nny \rightarrow ny+ny, ssz \rightarrow sz+sz, tty \rightarrow ty+ty$$
- The same for the double trigraph: $ddzs \rightarrow dzs+dzs$.
- The Polish alphabet is:

$$a < \acute{a} < b < c < \acute{c} < d < e < \acute{e} < \dots < l < \acute{l} < m < n < \acute{n} < o < \acute{o} < p < \dots < s < \acute{s} < t < \dots < z < \acute{z}$$
 - The Romanian alphabet is: $a < \check{a} < \hat{a} < b < \dots < i < \acute{i} < j < \dots < s < \check{s} < t < \acute{t} < u < \dots < z$.
 - The Slovak alphabet is:

$$a < \acute{a} < \check{a} < b < c < \check{c} < d < d' < dz < d\check{z} < e < \acute{e} < f < g < h < ch < i < \acute{i} < j < k < l < \acute{l} < \check{l} < m < n < \check{n} < o < \acute{o} < \hat{o} < p < q < r < \acute{r} < s < \check{s} < t < \acute{t} < u < \acute{u} < \dots < y < \acute{y} < z < \check{z}$$
 - The Spanish alphabet was $a < b < c < ch < d < \dots < l < ll < m < n < \check{n} < o < \dots < z$ until 1994. Now the digraphs ‘ch’ and ‘ll’ are no longer viewed as single letters in modern dictionaries, and the words using ‘ñ’ are interleaved with words using ‘n’.
 - In Swedish, accented letters are grouped at the end of the alphabet: $a < \dots < z < a < \check{a} < \check{o}$.

^a Using a left-to-right order for this step is common mistake even for French people. But the accurate order is right-to-left, as specified in [7].

^b *Deutsche Institut für Normung* (German Institute of normalisation).

Figure 1: Some order relations used in European languages.

could be easily updated for future versions dealing with Unicode.

In the first section, we show how diverse lexicographic orders used throughout European countries are. This allows readers to estimate this diversity and to realise the complexity of this task. We also explain why this problem is made more difficult when we consider it within the framework of bibliographies. Then we show how order relations operate in MIBIB¹TEX and how they are built. We also give some details about the common and different points between x^{indy} [13, § 11.3] and MIBIB¹TEX, both being programs using multilingual order relations. Reading this article does not require advanced knowledge of Scheme;² in fact, we think that a non-

² Readers can refer to [20] for an introductory book about Scheme.

programmer should be able to specify a new order relation. We give more technical details in an annex, for users that would like to experiment more themselves. In particular, we explain how to deal with languages using the Latin 2 encoding, despite our implementation being based on Latin 1.

1 European languages and lexicographic orders

Figure 1 gives an idea of the diversity of order relations used throughout some European countries. In this figure, ‘ $a < b$ ’ denotes that the words beginning with a are ‘less than’ the words beginning with b , whereas ‘ $a \sim b$ ’ expresses that the letters a and b are interleaved, except that a takes precedence over b if two words differ only by these two letters. Roughly speaking, there are two families of lan-

guages in the realm of associated lexicographic orders. Accented letters are sometimes fully viewed as ‘real’ letters, distinct from unaccented ones: examples are given by some Slavonic languages. In other languages, accented letters are sorted as if there were no accent. The precedence of a unaccented letter over an accented one is determined in various ways: it follows a left-to-right order in Irish, Italian, and Portuguese, a right-to-left order in French. The Estonian language ‘mixes’ the two approaches: some accented letters — ‘õ’, ‘ä’ — are alphabeticised, some — ‘š’, ‘ž’ — are interleaved. Last, some letter groups may be viewed as a single letter and alphabeticised as another letter. For example, the Hungarian words beginning with ‘cs’ are alphabeticised separately from the words beginning with ‘c’. In fact, the ‘c-’ entry in a Hungarian dictionary contains words beginning with ‘c’ and not with ‘cs’. The ‘c-’ entry is followed by the ‘cs-’ entry, before the ‘d-’ entry.

Anyway, it is apparent that there cannot be a universal order encompassing all lexicographic orders. Besides, these orders aim to classify words of a dictionary, that is, common words belonging to a language, even if some dictionaries may include some proper names. When bibliographies are generated, order relations are used to sort bibliographical items, most often w.r.t. authors’ names. These names may be ‘foreign’ proper names if we consider the language used for the bibliography. So names can include characters outside of this language’s alphabet. As a consequence, an order relation for sorting a bibliography should be able to deal with any letter, since any letter may appear in foreign names. A good choice is to associate such a foreign letter with a letter belonging to the ‘basic’ Latin alphabet, so this foreign letter is interleaved with the basic letter, which takes precedence over the foreign letter if two words differ only by these two letters. If we consider the English language, this means that accented letters are interleaved with unaccented letters, but unaccented letters take precedence. Most implementations of order relations proceed in this way.

Unicode provides a default algorithm to sort all its characters. This algorithm is based on a sort key table, DUCET³ [23]. It is also based on a decomposition property for composite characters. For example, the ‘ø’ letter, whose name and code point — given using hexadecimal numbers — are:

LATIN SMALL LETTER O WITH CIRCUMFLEX,
U+00F4

³ Default Unicode Collation Element Table.

can be decomposed into these ‘simpler’ characters:

LATIN SMALL LETTER O, U+006F
COMBINING CIRCUMFLEX ACCENT, U+0302

The sort algorithm requires several passes. To describe it roughly, an information about *weight*, given by *sort keys*, is associated with each string. Then this information is re-arranged according to sort levels, w.r.t. letters, w.r.t. accents, etc. Finally, a binary comparison between bytes is done, level by level, until the two strings can be distinguished. This algorithm can be refined for a particular language, by using a specialised sort key table, possibly including sort keys for accented letters and digraphs viewed as single letters. This *modus operandi* would be difficult to put into action within MIBIB_TE_X. First, we do not have complete support for Unicode⁴ for example, we cannot directly deal with characters such as the ‘combining circumflex accent’, not included in the Latin-1 encoding. But we keep the idea about decomposition, replacing the combining characters by ASCII⁵ characters. For example, the ‘combining circumflex accent’ will be replaced by the ‘~’ character. To sum up, our order relations are based on a 3-step algorithm:

- replace composite characters (‘foreign’ letters or composite characters not viewed as single letters) when extracting successive letter groups and compare the two results,
- refine the sort with accent information when accented letters are interleaved with others,
- test the case: when two words differ only in case, an uppercase letter takes precedence over a lowercase one, according to a left-to-right order.

2 Generating order relations

Let us recall that MIBIB_TE_X can apply BIB_TE_X’s bibliography styles using a compatibility mode [6], but in order to take advantage of MIBIB_TE_X’s multilingual features as far as possible, it is better to use the `nbst`⁶ language [4], close to XSLT⁷ [24], the language of transformations used for XML⁸ documents. Let us recall that parsing a bibliography data base (.bib) results in the representation of an XML tree in Scheme [11]; this `nbst` language includes an element for sorting selected subtrees of an XML document [4, App. A], this element being analogous to XSLT’s [24, § 10]. For example, the following two elements

⁴ See the annex.

⁵ American Standard Code for Information Interchange.

⁶ New Bibliography STyles.

⁷ eXtensible Stylesheet Language Transformations.

⁸ eXtensible Markup Language.

can be used to sort bibliographical items by the first author’s last name, and then the items left unsorted by this first step are sorted by the first author’s first name:⁹

```
<nbst:sort
  select="author/name[1]/personname/last"
  language="german"/>
<nbst:sort
  select="author/name[1]/personname/first"
  language="german"/>
```

Due to the `language` attribute’s value, this sort operation will use the lexicographic order for the German language. Such an order relation is to be specified in Scheme, as a 2-argument function taking two strings s_0 and s_1 and returning a ‘true’ value (`#t`) if s_0 is strictly less than s_1 , a ‘false’ value (`#f`) otherwise. The best way to define such a function is to derive it from a generator of order relations, as shown in Figure 2. This `<mk-order-relation` generator has four arguments.

- A list whose elements are *separator* characters, viewed as less than any letter. Usually, this list contains only the space character, in which case, the `<space-only` variable can be used. This is not universal: for example, space characters are ignored when words are sorted in Hungarian (cf. the definition of the `<hungarian?` variable in Figure 2).
- An alphabet, given w.r.t. the increasing order, as a list of strings. If the ‘classical’ alphabet is used — unaccented letters of the Latin alphabet, sorted according to the usual order — just put the ‘false’ value (cf. the definition of the `<english?` variable).
- An association list for additional sequences of characters, each sequence being followed by a replacement and a weight.
- A function related to the sense of the second step: when the first is finished and the second is about to start, weights appear in reverse order, so put **reverse!**¹⁰ (resp. **identity** — the identity function) to put the second step into action according to a left-to-right (resp. right-to-left) order. Cf. the use of these two values for `<french?` and `<english?`.

⁹ Let us notice that this illustrative example would be too restrictive for an ‘actual’ bibliography style: there may be several authors, and some authors may be denoted by an organisation name, in which case the element’s name is not `personname`, but `othername`.

¹⁰ Some Schemers could observe that this function does not belong to pure functional style, because it is potentially destructive [17]. But it is more efficient than the `reverse` function and the weight list is not shared with other lists.

It should be noted that only lowercase letters have to be specified, the equivalent relations among uppercase letters will be deduced.

Let us come back to associations for additional sequence characters. There are default associations, comparable to the information given by the decomposition property in Unicode. For example:

$$\acute{e} \mapsto e + |' |$$

where “|’|” denotes the default weight of the “ ’ ” character. MIBIB_{TEX} knows such decomposition information for each accented letter of Latin 1. These default associations can be overridden by alphabet-specific associations given to the function building orders. Weights are managed as follows.

- By default, the weight of each component of an alphabet — appearing within the second argument of `<mk-order-relation` — is 1.
- If we consider only one substitution, that is, a word \mathcal{W}_0 where a sequence \mathcal{S}_0 is to be replaced by a sequence \mathcal{S}_1 with a weight w_1 , this substitution resulting in a word \mathcal{W}_1 . The \mathcal{W}_0 word will be alphabeticised first if $w_1 < 1$, put after otherwise.

Here are some examples.

- In French, the only accent put on the ‘o’ letter is circumflex. When ‘ δ ’ is replaced by ‘o’ for the first step, we must ensure that ‘ δ ’ will be ranked after ‘o’ if two words differ only by these two letters at the same position. We must also ensure that the other accented letters based on ‘o’ — in ‘foreign’ words will be put after. So the weight of the replacement of ‘ δ ’ by ‘o’ is 2, as it can be seen in Figure 2 (cf. the definition of `<french?`). The default weights for accents are higher, so this accented letter is ranked before the other accented letters based on the ‘o’ letter and possibly used in languages other than French.
- Similarly, the two accents allowed on the ‘a’ letter are grave and circumflex, the correct order being $a < \grave{a} < \hat{a}$. So the replacement of ‘ \grave{a} ’ (resp. ‘ \hat{a} ’) by ‘a’ for the first step is 2-weight (resp. 3-weight).

Given a language, if a character belongs neither to separators, nor to the alphabet, it is ignored, unless it is an accented letter included in default associations.¹¹

Given an alphabet’s specification — the second argument of the `<mk-order-relation` function —

¹¹ As a consequence, some ‘exotic’ letters are ignored outside their own language, because they cannot be related to another letter of the Latin alphabet. For example, that is the case for the ‘p’ letter of the Icelandic language.

```

(define <english (<mk-order-relation <space-only #f '() reverse!))
(define <austrian?
  (<mk-order-relation
   <space-only
   '("a" "ä" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "ö" "p" "q" "r" "s" "t" "u"
     "ü" "v" "w" "x" "y" "z")
   '() reverse!))
(define <czech?
  (<mk-order-relation
   <space-only
   '("a" "b" "c" "\\v{c}" "d" "e" "f" "g" "h" "ch" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "\\v{r}"
     "s" "\\v{s}" "t" "u" "v" "w" "x" "y" "z" "\\v{z}")
   '() reverse!))
(define <danish?
  (<mk-order-relation
   <space-only
   '("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w"
     "x" "y" "z" "æ" "ø" "å")
   '(("aa" ("a" . 2))) ; In Danish, 'aa' is equivalent to 'a'.
   reverse!))
(define <estonian?
  (<mk-order-relation
   <space-only
   '("a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "z" "t" "u" "v"
     "w" "õ" "ä" "ö" "ü" "x" "y")
   '("\\v{s}" ("s" . 2)) ("\\v{z}" ("z" . 2))) reverse!))
(define <french?
  (<mk-order-relation <space-only #f
   '(("â" ("a" . 2)) ("â" ("a" . 3)) ("ê" ("e" . 2)) ("é" ("e" . 3))
     ("ê" ("e" . 4)) ("ë" ("e" . 5)) ("î" ("i" . 2)) ("ï" ("i" . 3))
     ("ö" ("o" . 2)) ("ü" ("u" . 2)) ("ü" ("u" . 3)) ("ÿ" ("y" . 2)))
   identity))
(define <german-din-1?
  (<mk-order-relation <space-only #f '(("ä" ("a" . 2)) ("ö" ("o" . 2)) ("ü" ("u" . 2))) reverse!))
(define <german-din-2?
  (<mk-order-relation
   <space-only #f '(("ä" ("a" . 2) ("e" . 2)) ("ö" ("o" . 2) ("e" . 2)) ("ü" ("u" . 2) ("e" . 2)))
   reverse!))
(define <hungarian?
  (<mk-order-relation
   '() ; In Hungarian, a space character is irrelevant when words are sorted.
   '("a" "b" "c" "cs" "d" "dz" "dzs" "e" "f" "g" "gy" "h" "i" "j" "k" "l" "ly" "m" "n" "ny" "o" "ö"
     "p" "q" "r" "s" "sz" "t" "ty" "u" "ü" "v" "w" "x" "y" "z" "zs")
   '(("á" ("a" . 2)) ("é" ("e" . 2)) ("ccs" ("cs" . 2)) ("cs" . 2))
     ("ddz" ("dz" . 2) ("dz" . 2)) ("ddzs" ("dzs" . 2) ("dzs" . 2)) ("ggy" ("gy" . 2) ("gy" . 2))
     ("i" ("i" . 2)) ("lly" ("ly" . 2) ("ly" . 2)) ("nny" ("ny" . 2) ("ny" . 2)) ("ó" ("o" . 2))
     ("\\H{o}" ("ö" . 2)) ("ssz" ("sz" . 2) ("sz" . 2)) ("tty" ("ty" . 2) ("ty" . 2))
     ("ú" ("u" . 2)) ("\\H{u}" ("ü" . 2)))
   reverse!))
(define <polish?
  (<mk-order-relation
   <space-only
   '("a" "\\aob" "b" "c" "\\{c}" "d" "e" "\\eob" "f" "g" "h" "i" "j" "k" "l" "\\l" "m" "n"
     "\\n" "o" "ó" "p" "q" "r" "s" "\\s" "t" "u" "v" "w" "x" "y" "z" "\\z")
   '() reverse!))

```

Figure 2: Building order relations for some European languages.

```

(define mk-hungarian-word-sectioner ; Building a generator of sectioning functions for Hungarian words.
  (<mk-otoken-generator '() ; The first three arguments of the <mk-order-relation
    '("a" "b" "c" "cs" ...) ; function in the definition of the <hungarian? variable:
    '(("á" ("a" . 2)) ...)) ; cf. Figure 2.

  (define g ; Definition of a zero-argument function that will
    (mk-hungarian-word-sectioner "sz\\H{o}l\\H{o}") ; section the word 'szőlő' ('grape').

  (g) => ("sz" . 1) ; The successive equivalent letters, digraphs, etc. of this word are returned in turn, with
  (g) => ("ö" . 2) ; the corresponding weight.
  (g) => ("l" . 1)
  (g) => ("ö" . 2)
  (g) => #f ; The word is finished, so all the calls of the g function will return the 'false' value, from now on.

```

Figure 3: How to section Hungarian words.

MIBIB_{TEX} notices the possible presence of multi-character sequences (e.g., digraphs or trigraphs). If need be, it builds a lexical analyser able to return the longest sequence of characters belonging to this alphabet,¹² an example of use being given in Figure 3. Let us mention that these analysers extract as few sequences of characters as possible. For example, if we have to compare a word beginning with ‘a’ and a word beginning with ‘b’ in English, only the first letters — “a” and “b” — are extracted because that is sufficient to determine the result.

Regarding the implementation, the encoding of the sequences of an alphabet w.r.t. an increasing order is implemented by means of *hash tables*,¹³ which ensures efficiency. Let us not forget that these order relations are used to sort bibliographical items, and sorting requires many calls to the function modelling an order relation.

3 MIBIB_{TEX} vs. xindy

xindy [9] and MIBIB_{TEX} do not aim to perform the same task, since xindy is an index processor. However, both have common points: they reimplement ‘old’ programs belonging to _{TEX}’s galaxy — *make-index* [13, § 11.2] and *BIB_{TEX}* — with a particular focus on multilingual features, they are both written using a Lisp¹⁴ dialect: Common Lisp [21] for xindy, Scheme for MIBIB_{TEX}. Of course, the successive steps used for putting an order relation into action — needed to arrange the successive entries of an index — also exist in xindy. But the specification

of an order relation is different because it is done step by step. There are forms:

```

define-alphabet      define-letter-group
merge-rule          sort-rule

```

to specify an alphabet, a letter group, and the replacement of a pattern. If a sort procedure is quite close to the standard way used in English, it is probably easier to use xindy’s forms, because only small changes have to be expressed. In MIBIB_{TEX}, we chose to develop fewer functions, which encapsulate the complete making of an order relation. This allows a global view of a new order relation and makes easier some coherence tests among the information about this relation.

4 Conclusion

The availability of these language-dependent order relations within a unique program has been planned through the use of the `language` attribute, as specified in the W3C¹⁵ recommendation about XSLT [24, § 10]. However, these relations have been implemented only partially in most of XSLT processors. Of course, our implementation also only partially provides this service, because we are limited to European languages. But we think that the orders we define are correct w.r.t. these languages and they are actually running. Our implementation is clearly influenced by the Unicode collation algorithm. It is a first step towards general algorithms for lexicographic orders, and a first version subject to changes when we explore other languages or get criticisms from end-users. In many domains, improvement has come about because first versions existed. We think that will be also the case for our functions.

5 Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has written the Polish translation of the abstract. I

¹² Such lexical analysers are implemented by means of *tries*. In MIBIB_{TEX}, this structure is also used to manage the information related to language identifiers, as explained in [5].

¹³ A *hash table* has a set of entries, and can efficiently map an object to another object. This structure is described in [1] from a general point of view, our implementation of hash tables in MIBIB_{TEX} is inspired by [8].

¹⁴ **LIS**T **P**ROCESSOR.

¹⁵ **W**ORLD **W**IDE **W**EB Consortium.

also thank Gyöngyi Bujdosó, Hans Hagen, Karel Horák, Dag Langmyhr, who helped me fix some errors. Thanks to Karl Berry and Barbara Beeton, who proofread the revised version.

A How to use MIBIB_TE_X's functions

A.1 Getting started

To use the functions dealing with multilingual ordering, change your current directory into the `src` subdirectory of MIBIB_TE_X's main directory, launch a Scheme interpreter, and proceed as follows:

```
(load "common.scm") ; Loading general
                      ; definitions.
(load "orders.scm") ; Loading all the
                    ; definitions related to orders. This causes
                    ; the other files needed to be loaded, too.
```

Then you can use the functions described in Figure 2. Use a R5RS-compliant Scheme interpreter [10] and one able to deal with the Latin 1 encoding: `bigloo` [16], MIT Scheme [3], and PLT Scheme [2] are suitable.¹⁶ There is also a file performing some tests: `tests/test-orders-unacc.scm`.

Now we describe the conventions used within strings resulting from parsing a `.bib` file. These conventions are supposed to be followed by the arguments of the functions modelling order relations, so you have to know them. You can directly type accented letters belonging to the Latin 1 encoding:

```
"Frank Böhmert"
```

In Scheme, the ‘`"`’ character being the delimiter of constant strings, it must be escaped by a ‘`\`’ character if it belongs to a string:

```
"\"Perry Rhodan\" Series"
```

If you are interested in strings using other encodings (in particular, the Latin 2 encoding, used in Eastern Europe), you cannot specify them directly; you must use the `ĀTEX` command producing accents and other diacritical signs not included in Latin 1. For example, ‘Henryk Mikołaj Górecki’ should be typed ‘`"Henryk Miko{\l}aj Górecki"`’ because ‘`ó`’ belongs to Latin 1, but ‘`ł`’ does not. Remember that the ‘`\`’ escape character must be itself escaped within a string. If such an accent command has no argument — e.g., the ‘`\l`’ command — write this command between braces, as suggested by the previous example. Use braces for the argument of an accent command, as in ‘`"Rezs{\H{o}} Kókai"`’ for ‘Rezső Kókai’.¹⁷

¹⁶ In fact, these three Scheme interpreters include partial support of Unicode, as mentioned in the introduction.

¹⁷ In fact, these letters belonging to the Latin 2 encoding are all defined as Scheme variables in the file `orders.scm`, e.g.:

Now you can type some expressions and evaluate them:

```
(<english? "coté" "côte") => #t ; True.
(<french? "coté" "côte") => #f ; False.
```

Of course, you can define new order relations according to the *modus operandi* we explain in § 2 and try to model some ‘exotic’ order relations.¹⁸

A.2 Testing decomposition

To see how words are sectioned into successive letters, digraphs, etc. according to a particular alphabet, then use the `<mk-otoken-generator` function to build a generator of functions sectioning words for a particular language. This `<mk-otoken-generator` function is automatically called when we apply the `<mk-order-relation` function, and its three arguments are the second, third and fourth arguments of the `<mk-order-relation` function. As an example, Figure 3 shows how to build and use such a generator for Hungarian words.

A.3 Going further

If you want to use MIBIB_TE_X for producing bibliographies — in which case you have to load more files by means of evaluating the expression:

```
(load "mlbibtex.scm")
```

— and would like to change the association of a language with an order relation, use such an expression:

```
(<c-language->order-relation
 "german"
 <german-din-2?) => #t
```

This causes `<-german-din-2?` to be the order relation used for German. If another relation was previously associated with this language,¹⁹ it is replaced by this new value, the `<-german-din-2?` function. If no order relation was known for this language,²⁰ the association is created. The result is `#t` if the association succeeds, `#f` otherwise (for example, a string whose value is an unknown language).

```
(define <l-slashed-string "{\l}")
```

```
(define <o-double-acute-string "\\H{o}")
```

... and used only by means of these variables. Of course, this complicates the definitions given in Figure 2, but when Scheme is Unicode-compliant, we will only have to change these definitions.

¹⁸ It can be noticed that all the names of the Scheme functions described above begin with ‘`<`’. A convention within the source files of MIBIB_TE_X is that all definitions made in the same file have the same prefix. That allows a ‘kind of modularity’, even if Scheme’s standard does not provide a way to emphasise modular decomposition. Of course, we recommend you choose a not-yet-used prefix for your own definitions.

¹⁹ In fact, when MIBIB_TE_X is initialised, the order relation for the German language is the `<german-din-1?` function.

²⁰ ... in which case the default order relation is the `<english?` function.

References

- [1] Alfred V. AHO, Ravi SETHI and Jeffrey D. ULLMAN: *Compilers, Principles, Techniques and Tools*. Addison-Wesley Publishing Company, 1986.
- [2] Matthew FLATT: *PLT MzScheme: Language Manual. Version 360*. August 2004. <http://download.plt-scheme.org/doc/360/pdf/mzscheme.pdf>.
- [3] Chris HANSON, THE MIT SCHEME TEAM *et al.*: *MIT Scheme Reference Manual*, 1st edition. March 2002. Massachusetts Institute of Technology.
- [4] Jean-Michel HUFFLEN: “MiBIB_TE_X’s Version 1.3”. *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.
- [5] Jean-Michel HUFFLEN: *Managing Languages within MiBIB_TE_X*. In revision. June 2005.
- [6] Jean-Michel HUFFLEN: “BIB_TE_X, MiBIB_TE_X and Bibliography Styles”. *Biuletyn GUST*, Vol. 23, pp. 76–80. In *BachoT_EX 2006 conference*. April 2006.
- [7] ISO-IEC CD 14651: *International String Ordering — Method for Comparing Character Strings and Description of a Default Tailorable Ordering*. May 1996.
- [8] Panu KALLIOKOSKI: *Basic Hash Tables*. September 2005. <http://srfi.schemers.org/srfi-69/>.
- [9] Roger KEHR: *xindy Manual*. February 1998. <http://www.xindy.org/doc/manual.html>.
- [10] 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: “Revised⁵ Report on the Algorithmic Language Scheme”. *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.
- [11] Oleg E. KISELYOV: *XML and Scheme*. September 2005. <http://okmij.org/ftp/Scheme/xml.html>.
- [12] Leslie LAMPORT: *L_AT_EX: A Document Preparation System. User’s Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.
- [13] Frank MITTELBACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L_AT_EX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [14] Oren PATASHNIK: *Designing BIB_TE_X Styles*. February 1988. Part of the BIB_TE_X distribution.
- [15] Oren PATASHNIK: *BIB_TE_Xing*. February 1988. Part of the BIB_TE_X distribution.
- [16] Manuel SERRANO: *Bigloo. A Practical Scheme Compiler. User Manual for Version 2.9a*. December 2006.
- [17] Olin SHIVERS: *List Library*. October 1999. <http://srfi.schemers.org/srfi-1/>.
- [18] Michael SPERBER, William CLINGER, R. Kent DYBVIG, Matthew FLATT, Anton VAN STRAATEN, Richard KELSEY and Jonathan REES: *Revised⁶ Report on the Algorithmic Language Scheme — Standard Libraries*. September 2007. <http://www.r6rs.org>.
- [19] Michael SPERBER, William CLINGER, R. Kent DYBVIG, Matthew FLATT, Anton VAN STRAATEN, Richard KELSEY, Jonathan REES, Robert Bruce FINDLER and Jacob MATTHEWS: *Revised⁶ Report on the Algorithmic Language Scheme*. September 2007. <http://www.r6rs.org>.
- [20] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.
- [21] Guy Lewis STEELE, JR., Scott E. FAHLMAN, Richard P. GABRIEL, David A. MOON, Daniel L. WEINREB, Daniel Gureasko BOBROW, Linda G. DEMICHIEL, Sonya E. KEENE, Gregor KICZALES, Crispin PERDUE, Kent M. PITMAN, Richard WATERS and Jon L WHITE: *COMMON LISP. The Language. Second Edition*. Digital Press. 1990.
- [22] THE UNICODE CONSORTIUM: *The Unicode Standard Version 5.0*. Addison-Wesley. November 2006.
- [23] THE UNICODE CONSORTIUM, <http://unicode.org/reports/tr10/>: *Unicode Collation Algorithm*. Unicode Technical Standard #10. July 2006.
- [24] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.

Introducing L^AT_EX users to XSL-FO*

Jean-Michel Huffle

LIFC (EA CNRS 4157)

University of Franche-Comté

16, route de Gray

25030 BESANÇON CEDEX

France

huffle (at) lifc dot univ-fcomte dot fr

http://lifc.univ-fcomte.fr/~huffle

Abstract

This talk aims to introduce L^AT_EX users to XSL-FO. It does not attempt to give an exhaustive view of XSL-FO, but allows a L^AT_EX user to get started. We show the common and different points between these two approaches of word processing.

Keywords L^AT_EX, XSL-FO, XML, XSLT.

Streszczenie

Prezentacja jest wprowadzeniem użytkowników L^AT_EX-a do XSL-FO. Nie próbujemy omówić XSL-FO w sposób wyczerpujący, ale umożliwimy użytkownikom L^AT_EX-a rozpoczęcie pracy w tej technologii. Pokażemy punkty wspólne i różnice obu podejść do formatowania tekstów.

Słowa kluczowe L^AT_EX, XSL-FO, XML, XSLT.

0 Introduction

This talk aims to introduce L^AT_EX users to XSL-FO.¹ Both have common points, in the sense that they are not WYSIWYG.² In both cases, users prepare a source file that is processed and the result is a file that can be sent to a laser printer. [11, §18] lists some implementations of processors of XSL-FO texts. Among them, we personally have experienced:

- **PassiveT_EX** [10, p. 180]: this is an (incomplete) adaptation of T_EX in order to process XSL-FO texts; the result may be a DVI³ or PDF⁴ file;
- **Apache FOP**⁵ [3], written in Java, is more complete; the result may be a PDF or PostScript file, with other formats also being possible.

XSL-FO is an XML⁶ format that aims to describe high-quality print outputs. As we will see, this format is very verbose, but it is not intended for

direct use. Usually, XSL-FO texts result from applying an XSLT⁷ stylesheet to an XML text, as we will see. Thus this approach clearly separates *presentation* and *contents*. An XML text specifies contents, an XSL-FO text specifies presentation. However, we begin with a text directly typed in XSL-FO to give the broad outlines of this language, then we show an example of an XSLT program that generates such a text. We end with some words about internationalisation. Reading this article requires only basic knowledge of XML and XSLT.

1 Getting started

1.1 Basic notions

The notion equivalent to a document class of L^AT_EX consists of a *page model*, an example being given in Figure 1. The page model here is very simple: only one page, specified by the `fo:simple-page-master` element. It specifies a paper format and its margins, where anything cannot be printed. It also defines *regions*, as shown in Figure 2. You can define *several* single page models, and another element, `fo:page-sequence-master`, allows the combination of single or repeatable pages. Repeatable pages may

* Title in Polish: *Wprowadzenie do XSL-FO dla użytkowników L^AT_EX-a*.

¹ **EX**tensible **St**ylesheet **L**anguage-**F**ormatting **O**bjects.

² **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et. This expression identifies typical interactive word processors.

³ **D**e**V**ice-**I**ndependent **F**ile.

⁴ **P**ortable **D**ocument **F**ormat.

⁵ **F**ormatting **O**bjects **P**rocessor.

⁶ **EX**tensible **M**arkup **L**anguage. Readers interested in an introductory book to this formalism can consult [12].

⁷ **EX**tensible **St**ylesheet **L**anguage **T**ransformations. Several introductory talks to this language have already been given at BachoT_EX conferences [4, 5]; the reference is [14].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:layout-master-set xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <!-- xmlns:fo declares a prefix for the namespace associated with XSL-FO texts. -->
  <fo:simple-page-master master-name="page-simple" page-height="297mm" page-width="210mm"
    margin-top="10mm" margin-bottom="20mm" margin-left="25mm"
    margin-right="25mm">
    <fo:region-before extent="30mm"/> <!-- Declaration of the header, footer, left and right -->
    <fo:region-after extent="30mm"/> <!-- margin. These usual terms have been viewed as too -->
    <fo:region-start extent="30mm"/> <!-- specific to left-to-right writing, thus a -->
    <fo:region-end extent="30mm"/> <!-- terminology based on 'before', 'after', 'start', 'end' -->
    <fo:region-body/> <!-- is preferred. The body is defined as the page's -->
    <!-- remainder. See Figure 2. -->
  </fo:simple-page-master>
</fo:layout-master-set>
```

Figure 1: Example of a page model in XSL-FO.

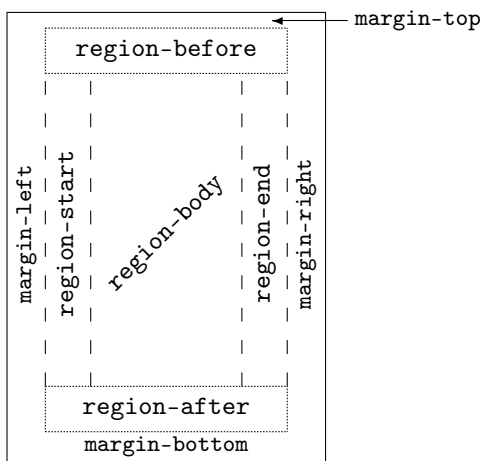
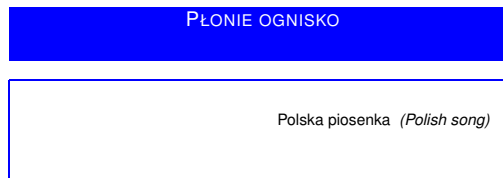


Figure 2: Regions defined by XSL-FO.



Płonie ognisko w lesie,
 Wiatr smętną piosnkę niesie.
 Przy ogniu zaś drużyna
 Gawędę rozpoczyna

Czuj, czuj, czuwaj,
 Czuj, czuj, czuwaj,
 Rozlega się dokoła,
 Czuj, czuj, czuwaj,
 Czuj, czuj, czuwaj,
 Najstarszy druh zawoła.

Przestańcie się już bawić
 I czas swój marnotrawić.
 Niechj każdy z was się szczerze,
 Do pracy swej zabierze

Czuj, czuj, czuwaj,
 Czuj, czuj, czuwaj,
 Rozlega się dokoła,
 Czuj, czuj, czuwaj,
 Czuj, czuj, czuwaj,
 Najstarszy druh zawoła.

vary w.r.t. the position, that is, you can alternate two models for even and odd pages, or define a separate model for initial and final pages, . . .

Figure 3 shows how an XSL-FO text may be formatted, the source text being given in Figure 4. We will see that page models are not specified by including a file as in L^AT_EX. If you wish a page model to be shared among several XSL-FO texts, an external entity is to be used [12, pp. 50–52]. This implies the introduction of a ‘dummy’ DOCTYPE tag.⁸ We see that an XSL-FO text is rooted by an fo:root element, whose children are a page model and a page sequence. A page sequence defines what is written and where. In Figure 4, a static content — a song’s title, followed by the number of the current page — is related to any page foot, whereas a flow allows the

Płonie ognisko (1)

Figure 3: The formatted output of Figure 4.

specification of a text possibly printed on regions belonging to several successive pages. A flow is bound to a region by means of the flow-name attribute, referring to the region-name attribute’s value of an element for a region. There are default conventions; for example, the definition of the ‘body’ region given in Figure 1 is equivalent to:

```
<fo:region-body
  region-name="xsl-region-body"/>
```

⁸ . . . which is a trick. A better method consists of using tags belonging to XInclude [15], but make sure that they are recognised by the tools you are using.

```

<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE root [<!ENTITY layout SYSTEM "layout.fo">
    <!ENTITY refren-1 "Czuj, czuj, czuwaj,">]>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  &layout;

  <fo:page-sequence master-reference="page-simple" font-family="serif" font-size="12pt"
    text-align="left">

    <fo:static-content flow-name="xsl-region-after">
      <fo:block text-align="center" line-height="14pt" color="green" font-size="10pt"
        font-family="serif">
        Płonie ognisko (<fo:page-number/>)
      </fo:block>
    </fo:static-content>

    <fo:flow flow-name="xsl-region-body" xml:lang="po">

      <fo:block font-family="sans-serif" font-size="18pt" font-variant="small-caps"
        padding-top="3pt" text-align="center" color="white" background-color="blue"
        space-after="15pt" line-height="24pt">
        Płonie ognisko
      </fo:block>

      <fo:block font-family="sans-serif" font-size="14pt" space-after="18pt" border-style="solid"
        border-width="0.5mm" border-color="blue" padding="4mm" start-indent="80mm"
        end-indent="4mm">
        <fo:block text-align="right">
          Polska piosenka
          <fo:inline font-style="italic" xml:lang="en">(Polish song)</fo:inline>
        </fo:block>
      </fo:block>

      <fo:block space-before.minimum="10pt" space-before.optimum="11pt"
        space-before.maximum="12pt">
        Płonie ognisko w lesie,
      </fo:block>

      <fo:block>Wiatr smętną piosnkę niesie.</fo:block>
      <fo:block>Przy ogniu zaś drużyna</fo:block>
      <fo:block>Gawędę rozpoczyna</fo:block>

      <fo:block ...> <!-- As above for the stanza's first line. -->
        &refren-1;
      </fo:block>
      <fo:block>&refren-1;</fo:block>
      <fo:block>Rozlega się dokoła,</fo:block>
      <fo:block>&refren-1;</fo:block>
      <fo:block>&refren-1;</fo:block>
      <fo:block>Najstarszy druh zawoła.</fo:block>

      <fo:block ...>Przestańciesię już bawić</fo:block>
      <fo:block>I czas swój marnotrawić.</fo:block>
      <fo:block>Niechj każdy z was się szczerze,</fo:block>
      <fo:block>Do pracy swej zabierze</fo:block>

      ... <!-- The refrain again. -->

    </fo:flow>

  </fo:page-sequence>
</fo:root>

```

Figure 4: Complete source of the text of Figure 3.

Attribute	Default value	Other values
<code>font-family</code>	<code>serif</code>	<code>sans-serif</code>
<code>font-size</code>		Absolute sizes: <code>xx-small</code> , <code>x-small</code> , <code>medium</code> , <code>large</code> , <code>x-large</code> , <code>xx-large</code> , Relative sizes: <code>smaller</code> , <code>larger</code> Lengths: e.g., <code>10pt</code>
<code>font-stretch</code>	<code>normal</code>	<code>wider</code> , <code>narrower</code> , <code>ultra-condensed</code> , <code>extra-condensed</code> , <code>condensed</code> , <code>semi-condensed</code> , <code>semi-expanded</code> , <code>expanded</code> , <code>extra-expanded</code> , <code>ultra-expanded</code>
<code>font-weight</code>	<code>normal</code>	<code>bold</code> , <code>bolder</code> , <code>lighter</code>
<code>font-style</code>	<code>normal</code>	<code>italic</code> , <code>reverse-normal</code> , <code>reverse-oblique</code>
<code>font-variant</code>	<code>normal</code>	<code>small-caps</code>

Table 1: Possible values for most of font attributes.

1.2 Formatting texts

At first glance, `fo:block` elements are analogous to paragraphs in L^AT_EX. The text inside a block may be justified, left or right aligned, according to the value of `text-align`. The attributes `color` and `background-color` specify colours for the text and background. Other attributes:

`border-style` `border-width` `border-color`

allows us to draw a box around this block. Of course, `border-width` is set to a null value by default, so no border is drawn. The ‘padding-...’ attributes characterise the padding between the text and border [10, pp. 96–100].

Blocks may be nested and most attributes are inherited. As an example, let us consider the second block of the flow. It defines some attributes related to fonts — `font-family` and `font-size` — these attributes being inherited in the nested block containing the Polish and English words for ‘Polish song’ (*‘Polska piosenka’*). The `fo:inline` element allows some attributes to be redefined without opening a new block: it corresponds to changing some parameters — font style or size, etc. — inside the same paragraph in L^AT_EX. In fact, we can consider that `fo:block` elements, due to this recursive nature, are equivalent to both the `\par` command and the `minipage` environment of L^AT_EX. The possible values associated with most of the font attributes are given in Table 1. In comparison with L^AT_EX where the family, weight, style, and variant of a font are expressed by combinations of commands being the same syntax, ‘`\text...{...}`’, the attributes of XSL-FO are more ‘typed’. That may be seem quite artificial to a L^AT_EX user, but emphasises all the possible combinations.

The `start-indent` attribute specifies the distance from the start-edge of the box surrounding the contents to the start-edge of the contents itself. The `end-indent` attribute is analogous, but end-edges are considered. The vertical spacing between

successive blocks is controlled by the two attributes `space-before` and `space-after`. The specification of *stretcheable lengths* in L^AT_EX [7, § A.1.15] is implemented in XSL-FO by means of *components*. Let us look at the first stanza given in Figure 4: the vertical spacing before this block is ideally 11 pt long, at least 10 pt long, and at most 12 pt long, according to the values of the components `optimum`, `minimum`, and `maximum` of the `space-before` attribute. Just specifying `space-before="11pt"` sets the three components of the `space-before` attribute to this length. Putting:

```
space-before="11pt"
space-before.minimum="10pt"
```

only redefines the `minimum` component, the two others being 11 pt long.

Going thoroughly into this notion, XSL-FO provides two other components for the specification of spacing. The `conditionality` component controls whether a space-specifier has effect at the beginning or end of a reference area — e.g., the beginning (resp. end) of a page for the `space-before` (resp. `space-after`) attribute of the `fo:block` element, or the beginning (resp. end) of a line for the `space-start` (resp. `start-end`) attribute of the `fo:inline` element. The possible values for this `conditionality` component are `discard` (by default) and `retain`. The `precedence` component can either be an integer or the keyword `force`. It determines what happens when the end of a reference area conflicts with the next one. If the `precedence` component is set to `force`, this will override any other space specifiers that conflict with it.

Let us briefly mention two attributes for blocks or inline texts: `text-decoration` is used to draw a line above, below, or through a text [16, § 7.17.4], `baseline-shift` is used for subscripts and superscripts. Since XSL-FO only aims to give nice layout of a text, there is no practical way to do computations on this text. For example, the fragment:

```
\iflanguage{polish}{Polska piosenka}{%
  Polish song}
```

(cf. [7, §9.2.1] about the `\iflanguage` command) cannot be transcribed into an XSL-FO text. However, some typical transformations can be put into action by means of the `text-transform` attribute, whose values may be `none` (by default), `capitalize`, `uppercase`, `lowercase`. Let us notice that using this attribute is somewhat deprecated because these operations do not make sense given internationalisation issues.

Other attributes prevent the breaking of a text into lines, columns, and pages when blocks are typeset: `keep-with-next`, `keep-with-previous`, and `keep-together`. Each of these three attributes has three components: `within-line`, `within-column`, and `within-page`. The associated values are `auto` (by default), that is, no constraint, `always`, or an integer expressing the strength of this property. This integer can be compared to the optional argument of the L^AT_EX commands `\pagebreak` and `\linebreak`. For example, if there is a `fo:block` element with a `keep-with-next` attribute set to `always`, there cannot be a page break between this block and the preceding one. If you want to force breaking in such situations, use the attributes `break-before` and `break-after`, whose values are `auto` (by default), `column`, `page`, `even-page`, and `odd-page`. See [10, pp. 70–72] for more details.

1.3 Additional elements

Now we mention some additional functionalities of XSL-FO, in order to give an idea of its expressive power. It provides elements to express lists, analogous to L^AT_EX's, rooted by the `fo:list-block` element [10, pp. 102]. The way to specify tabulars is analogous to HTML's,⁹ the most commonly used element for this being `fo:table` [10, pp. 104–110]. Footnotes are specified via the `fo:footnote` element [10, pp. 154–155], analogous to the `\footnote` command. Cross references as in L^AT_EX are supported by means of the `fo:basic-link` element [10, pp. 146–148]; hyper-link references to external documents are also possible. The notion of floating objects is known within XSL-FO: see [16, §6.12.2] about the `fo:float` element. The language provides elements and attributes for building indexes [16, §7.24], analogous to what is used within L^AT_EX's `theindex` environment (cf. [7, §11.1]). Last, let us notice that there is no mathematical mode in XSL-FO.

⁹ HyperText Markup Language. Readers interested in an introduction to this language can refer to [9].

2 XSLT and XSL-FO together

The Polish song given in Figure 4 has already been specified as a 'pure' XML text in [6, Fig. 1]. We reproduce it as Figure 5. Then we give an XSLT stylesheet (Figures 6 and 7) that yields the text of Figure 4 when it is applied to the XML text of Figure 5. That shows how XSL-FO texts should be built: by derivation from XML texts by applying a stylesheet.

The use of two namespaces [12, pp. 41–45] given by prefixes, `xmlns:xsl` and `xmlns:fo`,¹⁰ clearly separates what is evaluated (`<xsl:.../>`) when the XSLT program is running, and what results from this operation (`<fo:.../>`). Finally, let us notice that XSL-FO does not provide a way to build a table of contents automatically, but doing this task is easy when an XSLT program is used [10, pp. 149–150].

3 Some words on internationalisation

XSL-FO provides properties — that is, attributes — for specifying hyphenation properties [16, §7.10]. These attributes includes the specification of a country, a language, a hyphenation character, etc. In practice, the predefined attribute `xml:lang` — see the two occurrences of this attributes in Figure 4 — is treated as a shorthand and used to set the country and language properties [16, §7.31.24]. This attribute characterizes the language of a content by a two-letter language, optionally followed by a two-letter country code, as specified in [1].

XSL-FO is not limited to languages using the Latin alphabet and can deal with any writing mode. The `writing-mode` attribute can be set to:

- `lr-tb`, for 'left-to-right, top-to-bottom' (by default),
- `rl-tb`, for 'right-to-left, top-to-bottom',
- `tb-rl`, for 'top-to-bottom, right-to-left',
- or others [16, §7.29.7].

This specifies two directions: the first is the inline-progression-direction which determines the direction in which words will be placed and the second is the block-progression-direction which determines the direction in which blocks and lines are placed one after another. The inline-progression-direction for a sequence of characters may be implicitly determined using bidirectional character types for those characters from the Unicode Character Database [13] and the Unicode bidirectional algorithm [13, Annex 9].

¹⁰ In fact, the information identifying a precise namespace is not the prefix itself, but the value associated with it, e.g., `'http://www.w3.org/1999/XSL/Transform'` for an XSLT program. `XInclude` (see Footnote 8, p. 110) introduces another namespace to model file inclusions [15].

```

<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE poem0 SYSTEM "poem0.dtd" [

```

Figure 5: Example of a Polish song as an XML text.

4 Going further

Of course, we have not shown all the features of XSL-FO; our goal was merely to show that the basic features are analogous to L^AT_EX's, even if methods for advanced features diverge. We hope you are now able to write simple texts in XSL-FO. If you wish to go thoroughly into learning it, the reference is the W3C¹¹ recommendation of the latest version (1.1) [16]. [10] is more didactic, but is based on XSL-FO's Version 1.0, although the differences are very slight for simple examples. [2, ch. 8] and [8] are very didactic, too, and may be of interest for French- or German-speaking people, but have the same drawback.

5 Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who wrote the Polish translation of the abstract. I also thank

Karl Berry and Barbara Beeton, who proofread the definitive version.

References

- [1] Harald Tveit ALVSTRAND: *Request for Comments: 3066. Tags for the Identification of Languages*. UNINETT, Network Working Group. March 1995. <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc3066.html>.
- [2] Bernd AMMAN et Philippe RIGAUX : *Comprendre XSLT*. Éditions O'Reilly France. Février 2002.
- [3] *Apache FOP*. January 2007. <http://xmlgraphics.apache.org/fop/>.
- [4] Jean-Michel HUFFLEN: "Introduction to XSLT". *Biuletyn GUST*, Vol. 22, pp. 64. In *BachoT_EX 2005 conference*. April 2005.
- [5] Jean-Michel HUFFLEN: "Advanced Techniques in XSLT". *Biuletyn GUST*, Vol. 23, pp. 69–75.

¹¹ World Wide Web Consortium.


```

<?xml version="1.0"?>
<!DOCTYPE stylesheet [

```

Figure 6: An XSLT stylesheet that transforms the source given in Figure 5 to Figure 4..

```

<xsl:template name="put-footer">
  <xsl:param name="the-string"/>
  <fo:static-content flow-name="xsl-region-after">
    <fo:block text-align="center" line-height="14pt" color="green" font-size="10pt"
      font-family="serif">
      <xsl:value-of select="concat($the-string, ' (')"/><fo:page-number/><xsl:text></xsl:text>
    </fo:block>
  </fo:static-content>
</xsl:template>

<xsl:template name="put-title">
  <xsl:param name="the-title"/>
  <fo:block font-family="sans-serif" font-size="18pt" font-variant="small-caps"
    padding-top="3pt" text-align="center" color="white" background-color="blue"
    space-after="15pt" line-height="24pt">
    <xsl:value-of select="$the-title"/>
  </fo:block>
  <fo:block font-family="sans-serif" font-size="14pt" space-after="18pt" border-style="solid"
    border-width="0.5mm" border-color="blue" padding="4mm" start-indent="80mm"
    end-indent="4mm">
    <fo:block text-align="right">
      <xsl:value-of select="concat($polish-song-po, ' ')/>
      <fo:inline font-style="italic" xml:lang="en">
        <xsl:value-of select="concat('(', $polish-song-en, ')')"/>
      </fo:inline>
    </fo:block>
  </fo:block>
</xsl:template>
</xsl:stylesheet>

```

Figure 7: XSLT program of Figure 6, continued.

- In *BachTeX 2006 conference*. April 2006.
- [6] Jean-Michel HUFFLEN: “Writing Structured and Semantics-Oriented Documents: \TeX vs. XML”. *Biuletyn GUST*, Vol. 23, pp. 104–108. In *BachTeX 2006 conference*. April 2006.
- [7] Frank MITTELBACH, Michel GOOSSENS and Johannes BRAAMS, with David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L^AT_EX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [8] Manuel MONTERO PINEDA und Manfred KRÜGER: *XSL-FO in der Praxis. XML-Verarbeitung für PDF und Druck*. 2004.
- [9] Chuck MUSCIANO and Bill KENNEDY: *HTML & XHTML: The Definitive Guide*. 5th edition. O’Reilly & Associates, Inc. August 2002.
- [10] Dave PAWSON: *XSL-FO*. O’Reilly & Associates, Inc. August 2002.
- [11] Sebastian RAHTZ: “The TEI/ \TeX Interface”. In: *Proc. EuroTeX 2005*, pp. 38–49. Pont-à-Mousson, France. March 2005.
- [12] Erik T. RAY: *Learning XML*. O’Reilly & Associates, Inc. January 2001.
- [13] THE UNICODE CONSORTIUM: *The Unicode Standard Version 5.0*. Addison-Wesley. November 2006.
- [14] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [15] W3C: *XML Inclusions (XInclude) Version 1.0*, 2nd edition. W3C Recommendation. Edited by Jonathan Marsh, David Orchard, and Daniel Veillard. November 2006. <http://www.w3.org/TR/2006/REC-xinclude-20061115/>.
- [16] W3C: *Extensible Stylesheet Language (XSL). Version 1.1*. W3C Recommendation. Edited by Anders Berglund. December 2006. <http://www.w3.org/TR/2006/REC-xs11-20061205/>.

Using T_EX in a wiki

Tomasz Łuczak

Katowice, Poland

tlu (at) technodat dot com dot pl

<http://team-tl.livenet.pl>

Abstract

This article describes the use of wikis as sources and presentations of texts, with T_EX as a hidden engine for typesetting the wiki content.

1 The preliminaries

When one wants both a printed version of a document (usually in PDF format) and in the form of a web page, one usually starts from a T_EX document, which is converted to HTML. However, this requires that the author has at least basic T_EXnical skills. If several people who do not know anything about T_EX work on a document and the printed version is the last step in the chain, then starting from a web page might be a more appropriate direction.

The process of creating technical documentation thus can involve the editing of web pages — we have an instant presentation of the current state of the document. The next step is conversion to the T_EX format followed by compilation. It is important to note that the converted documents should not need to be manually adjusted or corrected.

2 The need

One of the practical uses for such a “from web to T_EX” process is creation of a system’s quality management documentation. The documentation is being created by employees who usually have no T_EXnical skills. This leads to the idea of providing such tools with which they are familiar, but of course we will not accept any loss of quality of the typeset result.

3 The execution

If one insists on ease of editing combined with keeping a change history, as well as the ease of converting into T_EX, it turns out that a wiki is the simplest solution. We only have to choose the proper program.

Dokuwiki was used for this project (<http://wiki.splitbrain.org/wiki:dokuwiki>), thanks to its automatic table of contents generation and page fragments editing. *Dokuwiki* keeps pages UTF-8 encoded without any other control codes. This enor-

mously simplifies the realization of an automatic converter.

To restrict authors’ invention, a set of “wiki complementing rules” was prepared which listed the allowed markup tags along with examples of use. During the preparation of these “rules” it turned out that the following features suffice: text emphasis; tables; drawings; itemization and enumeration without nesting; and headings for three levels (chapters, sections and subsections).

Unfortunately, the wiki does not automatically number chapters or sections, so to achieve a uniform look we decided to include chapter and section numbers directly into the titles. This solution is not a most elegant, but proved to be effective. Incidentally, this also gave section numbers in the automatically generated table of contents for a given wiki page.

Limiting the allowed tags made writing the converter easy. The code written in TCL fits into about hundred lines, mostly thanks to the following limitations: the PDF document is only for printing so the wiki’s hyperreferences were not converted; all tables had the same structure; and similarly, all drawings had the text column width.

Due to the deficiency of the converter authors were required to put an empty line to mark the end of a list. Unfortunately this was not always adhered to, leading to the biggest issue during conversion and compilation.

4 The summary

The use of the wiki and a web browser proved to be a hit: documents were quickly and willingly edited by the authors. The documentation not being up-to-date proved not to be an issue. Last but not least: the readers liked the easy to read and nicely typeset documents.

Single-source publishing in multiple formats for different output devices*

Petr Sojka

Faculty of Informatics, Masaryk University
Botanická 68a, 60200 Brno, Czech Republic
sojka (at) fi dot muni dot cz
<http://www.fi.muni.cz/usr/sojka/>

Michal Růžička

Faculty of Informatics, Masaryk University
Botanická 68a, 60200 Brno, Czech Republic
xruzick7 (at) fi dot muni dot cz

Abstract

TeX is traditionally used as an authoring tool for paper publishing of scientific texts and textbooks. Parallel electronic publications that are meant for on-screen viewing and web delivery are demanded by readers for many reasons today. The paper discusses ways to single-source author publishing from a L^AT_EX source file, and shows examples of several textbooks published by this approach. Special attention is given to the web document generation either to HTML or XHTML markup with the notation translated to MathML.

On-the-fly personalised document generation with JBIG2-compressed pictures for a digital library project DML-CZ is discussed as well.

1 Motivation

*Discover the outer logic of the typography
in the inner logic of the text.*
— Robert Bringhurst [6]

Documents conveying information have their *content* and *form*. Form (appearance) should reflect the *design*, it should use the graphical means consistently. Possibilities of a form of a document are constrained by an *output device* (paper, LCD monitor, PDA).

It is a well-known, but little respected fact, that the design of a document has to be (re)done for every new output device. Many documents fine-tuned with our TeX-based systems for reading on a paper (microtypography etc.) are often proudly posted on the web without redoing the design phase for a particular output device (LCD screen, PDA), for a particular purpose, or for specific readers' requests. This is in contrast with the goal Knuth had in mind e.g. when designing the fonts in METAFONT: even the tiny rasterization details affected by different printers should be fine-tuned by proper settings for a particular printer in `modes.mf`.

The authors who use open-sourced TeX-based system have significant power and possibilities to

influence every detailed aspect of the form when writing their papers and books. In the case when authors write using logical markup only, it is then possible to choose a different typography that honors content independently to suit different output device qualities by changing the design mapping for logical entities in the text. Strict separation of content and form is almost always possible, with only rare exceptions such as typesetting Christian Morgenstern's poems. However, authors must discipline themselves not to use visual typesetting commands such as `\vskip`.

As the quality and size of display devices grow, even longer texts will be read directly on screen, or document snippets on the XHTML browsers in mobiles or PDAs. Authors naturally demand that their content be ready for the wide range of different output devices. Although the design is inherently harder as usually the whole class of output devices and browsers has to be thought of, it is what is demanded by the readers.

In this paper we comment on several publishing projects: several textbooks [2, 3, 4] and database publishing in DML-CZ [11]. In all projects benefits from strict separation of form and content to produce different products are shown, using the single-source input created by the author or generated from a database on demand for different types of output.

* Support from the Czech Academy of Sciences grants AV1ET208050401 and AV1ET200190513 is acknowledged, as well as travel support from the Czechoslovak TeX Users Group.

2 Single-source publishing

*If the only tool we know is a word processor,
everything looks like a print document.*

— Peter Meyer [10]

An author wants to convey information, and the only thing she or he has to do is mark logical entities in the text. The designer should enforce sameness: visual rendering of the same logical parts should be consistently the same. This will allow publishing for different output devices just by switching between different designs. Maintenance of the text by the author will be much easier and cheaper than maintaining forked versions for different purposes. It will also reduce errors, improve consistency and/or save translation costs: the terms *single-source publishing* or *single-sourcing* are used for this type of document authoring (see http://en.wikipedia.org/wiki/Single_source_publishing). DocBook is a well-known system and DTD allowing single-source publishing, with support for conversion to XHTML, DVI, PostScript or PDF, with either XSL, XSL-FO or L^AT_EX.

Many single-source publishing approaches start from XML as the source of content [10]. For many authors, it is simply not convenient to write directly in XML, even with clever XML editors. Technical manuscripts full of mathematics will remain authored in some flavor of T_EX for the compactness and clarity of T_EX math notation. Author productivity raises significantly with author-centric system [9].

In academia, authors write textbooks for their courses, and want to publish them in the formats their students prefer. We have prepared two single-source textbooks [2, 4] from sources originally written in L^AT_EX. In the next sections we describe our experience with the project.

3 Markup and conversion tools

*Data cannot be used at a finer grain
than it is marked up at.* — Rick Jelliffe

To allow single-source publishing, we had to clean the source files significantly, as they were not written from the beginning with the goal to publish in different formats. Even DEK writes very “low-level” code in *The T_EXbook*:

```
&\elevenit I\kern.7ptillustrations by\cr
&DU\kern-1ptANE BIBBY\cr
\noalign{\vfill}
&\setbox0=\hbox{\manual77}%
\setbox2=\hbox to\wd0{\hss\manual6\hss}%
\raise2.3mm\box2\kern-\wd0\box0\cr % A-W logo
&ADDISON\kern.1em--WESLEY\cr
&PUBLISHING COMP\kern-.13emANY\kern-1.5mm\cr
```

He did not expect the code to be used for any-

thing else than printing to a phototypesetter, with the given fonts, kerning for given sizes, etc.

For single-source publishing, the main text has to be written without fine-tuning for a single output device or printer, and low-level markup has to be substituted by high-level commands allowing multiple macro definitions for different outputs. Markup must be written at the finest grain possible, expanding to the appropriate design setting for every type of output. The idea is not new, and it is used in the XML world as well (different CSS rendering for different devices or even browsers).

We have identified several types of output format our students have demanded. In addition to the standard version suitable for printing on paper, a searchable version optimised for an LCD screen with 4:3 aspect ratio was requested. For some purposes an (X)HTML version was needed for platforms and devices without a PDF renderer. Finally, we prepared an XHTML+MathML version as well.

There are many tools and utilities to convert T_EX documents to different output formats—a list of them is compiled on the T_EX Users Group web page *TeX Resources on the Web* (<http://www.tug.org/interest.html>). PDFL^AT_EX with the *hyperref* and *crop* packages is a suitable combination for print output. For on-screen versions of documents we have chosen the *pdfscreen* package in combination with pdfT_EX and the *hyperref* package.

4 PDF versions

*Make all visual distinctions as subtle as possible,
but still clear and effective.* — Edward R. Tufte [14]

For every output version, design parameters and macros are, as usual, written in separate conditional branch. A simple source code example:

```
\newif\ifprint
\printfalse % Non-print version.
%\printtrue % Print version.

\ifprint
  \hypersetup{colorlinks=false,
              pdfborder={0 0 0}}
  % Center mirrored document pages on A4
  % paper and add crop marks.
  \usepackage[cam,a4,center,mirror]{crop}
\fi
```

4.1 PDF for printing

For print it is desirable to prepare grey-scale output. For this job the *hyperref* package is often used with appropriate options (`colorlinks=false`, `pdfborder={0 0 0}`).

The *crop* package (available on CTAN) is a good choice to set up crop marks. The *crop* package is also

able to perform document transformation such as page mirroring, etc. The output is shown in Figure 1 on page 121.

4.2 PDF for on-screen viewing

A typical LCD screen is quite a different output device. Compared to current 1200 DPI printers, it has an order of magnitude lower resolution. Usually, it has a different aspect ratio, and allows colours of high depth. We can add interactive content such as hypertext links, navigation toolbar, etc.

To make versions of textbooks designed for the screen, we started with the package *pdfscreen* (available on CTAN). *pdfscreen* together with definitions of environments and macros for screens are again in a conditional branch of the textbook style file.

To save fine-tuning of line-breaking, we have made the line-breaking the same in both print and screen versions. Page breaking will be different, of course, as seen in Figure 2 on page 121.

5 Possibilities for web delivery

A buzzword today is XML—people spend much of their time browsing (X)HTML pages. For searchable and scalable math using outline fonts MathML is needed. There are several tools available:

- \TeX 2page (<http://www.ccs.neu.edu/home/dorai/tex2page/>),
- Tralics (<http://www-sop.inria.fr/apics/tralics/>),
- \TeX 4ht (<http://www.cse.ohio-state.edu/~gurari/TeX4ht/>),
- \LaTeX ML (<http://dlmf.nist.gov/LaTeXML/>)
- \LaTeX 2HTML (<http://latex2html.org>).

Each tool has some advantages and disadvantages; we do not want to discuss all of them. After testing some of the tools, we have chosen \TeX 4ht [5, 7]. \TeX 4ht uses the native \TeX compiler and processes the document with a special setup into standard DVI output with markup in `\specials`—there is no danger of omission of unsupported markup as in, e.g., \LaTeX 2HTML. From this enriched DVI, HTML pages are extracted by \TeX 4ht scripts.

5.1 HTML

The most difficult part of the conversion setup process was web output. In the case of complex documents it was usually necessary to make changes in the source code. These modifications and \TeX 4ht-specific commands are, of course, in a separate style file whenever possible.

Once we have source code in the \LaTeX format

with \TeX 4ht styles loaded we can try to convert the document. The first trial is to run the command:

```
htlatex filename.tex 'html'
```

When \TeX 4ht successfully completes its work we will get an HTML document with complicated formulae rendered into PNG images, as seen in Figure 3 on page 122. This way is simple and safe for rendering in all, even old, web browsers.

By default, the whole document is written to one HTML file, but \TeX 4ht is able to split a longer document into a tree of web pages automatically. Running `htlatex filename.tex 'html,2'` generates a document that has each chapter in a separate file. Navigation between chapters is available via a toolbar at the top and bottom of every page.

When generating HTML output it may be useful to insert some HTML code with the command `\HCode{Some HTML code.}`. This command can be used for CSS code insertion, too. \TeX 4ht offers both HTML and XHTML output generation.

We use existing \LaTeX logical markup as much as possible. For CSS code, we use the command `\Css{CSS definition}`. CSS attributes are mapped onto document elements through appropriate `\HCode` commands. The command `\ConfigureEnv` makes it possible to add our own code before and after environment content in the output document.

As a bit more complex example of CSS, this definition could serve for the theorem environment:

```
\newtheorem{theorem}{Theorem}[chapter]
...
\ifweb % In case of web format output...
  \Css{% CSS code definition block
    .theorem {
      background-color: \#FFFFFF;
      border: 1px solid;
      border-color: \#0000FF; } }
% In the resulting HTML document markup
% "<div class="theorem">" is placed
% before each "theorem" environment
% and markup "</div>" after that.
\ConfigureEnv{theorem}
  {\ifvmode \IgnorePar\fi
  \EndP\HCode{<div class="theore">}}
  {\ifvmode \IgnorePar\fi
  \EndP\HCode{</div>}}
  {}{}
\fi
...
% In the document, the same LaTeX markup is
% used for all versions: "theorem" environment
\begin{theorem}
  Function~$f$ have only one limit
  in point~$[x_{0},y_{0}]$.
\end{theorem}
```

18 *Pojem funkce více proměnných*

Pro $n = 2$ budeme místo $f(x_1, x_2)$ psát $f(x, y)$ a pro $n = 3$ místo $f(x_1, x_2, x_3)$ píšeme $f(x, y, z)$.

Příklad 1.1. i) Zobrazte v rovině definiční obor funkce

$$f(x, y) = \sqrt{\left(x^2 + \frac{(y-2)^2}{4} - 1\right)(x^2 + y^2 - 6x)}.$$

Řešení. Výraz pod odmocninou musí být nezáporný, tj. musí být splněna podmínka

$$\left(\frac{(y-2)^2}{4} + x^2 - 1\right)(x^2 + y^2 - 6x) \geq 0.$$

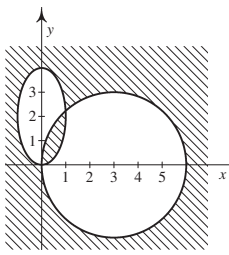
To nastane, právě když

$$\frac{(y-2)^2}{4} + x^2 - 1 \geq 0 \quad \text{a} \quad (x^2 + y^2 - 6x) \geq 0$$

nebo

$$\frac{(y-2)^2}{4} + x^2 - 1 \leq 0 \quad \text{a} \quad (x^2 + y^2 - 6x) \leq 0.$$

Rovnice $\frac{(y-2)^2}{4} + x^2 = 1$ je rovnicí elipsy se středem v bodě $[0, 2]$ a poloosami délek $a = 1$ a $b = 2$, rovnice $x^2 + y^2 - 6x = 0$ je rovnicí kružnice se středem v bodě $[3, 0]$ a poloměrem $r = 3$, neboť tuto rovnici lze převést na tvar $(x-3)^2 + y^2 = 9$. Množina všech bodů $[x, y] \in \mathbb{R}^2$ splňující výše uvedené nerovnosti, tj. definiční obor funkce f , je znázorněna na vedlejším obrázku. Je to uzavřená množina v \mathbb{R}^2 .



ii) Zobrazte v rovině definiční obor funkce

$$f(x, y) = \arccos(x^2 + y^2 - 1) + \sqrt{|x| + |y| - \sqrt{2}}.$$

Řešení. Definičním oborem funkce \arccos je interval $[-1, 1]$, první sčítanec je tedy definován pro $[x, y]$ splňující nerovnosti

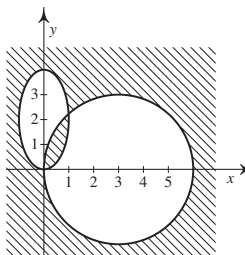
$$-1 \leq x^2 + y^2 - 1 \leq 1,$$

Figure 1: Print output (without page mirroring)

To nastane, právě když

$$\frac{(y-2)^2}{4} + x^2 - 1 \geq 0 \quad \text{a} \quad (x^2 + y^2 - 6x) \geq 0$$

nebo

$$\frac{(y-2)^2}{4} + x^2 - 1 \leq 0 \quad \text{a} \quad (x^2 + y^2 - 6x) \leq 0.$$


obr. 1.1 Definiční obor funkce f

Rovnice $\frac{(y-2)^2}{4} + x^2 = 1$ je rovnicí elipsy se středem v bodě $[0, 2]$ a poloosami délek $a = 1$ a $b = 2$, rovnice $x^2 + y^2 - 6x = 0$ je rovnicí kružnice se středem v bodě $[3, 0]$ a poloměrem $r = 3$, neboť tuto rovnici lze převést na tvar $(x-3)^2 + y^2 = 9$. Množina všech bodů $[x, y] \in \mathbb{R}^2$ splňující výše uvedené nerovnosti, tj. definiční obor funkce f , je znázorněna na obrázku 1.1. Je to uzavřená množina v \mathbb{R}^2 .

Titulní strana

Obsah

Výsledky cvičení

Rejstřík

◀ ▶

◀ ▶

Strana 21 z 451

Zpět

Vpřed

Zavřít

Konec

Figure 2: Screen output

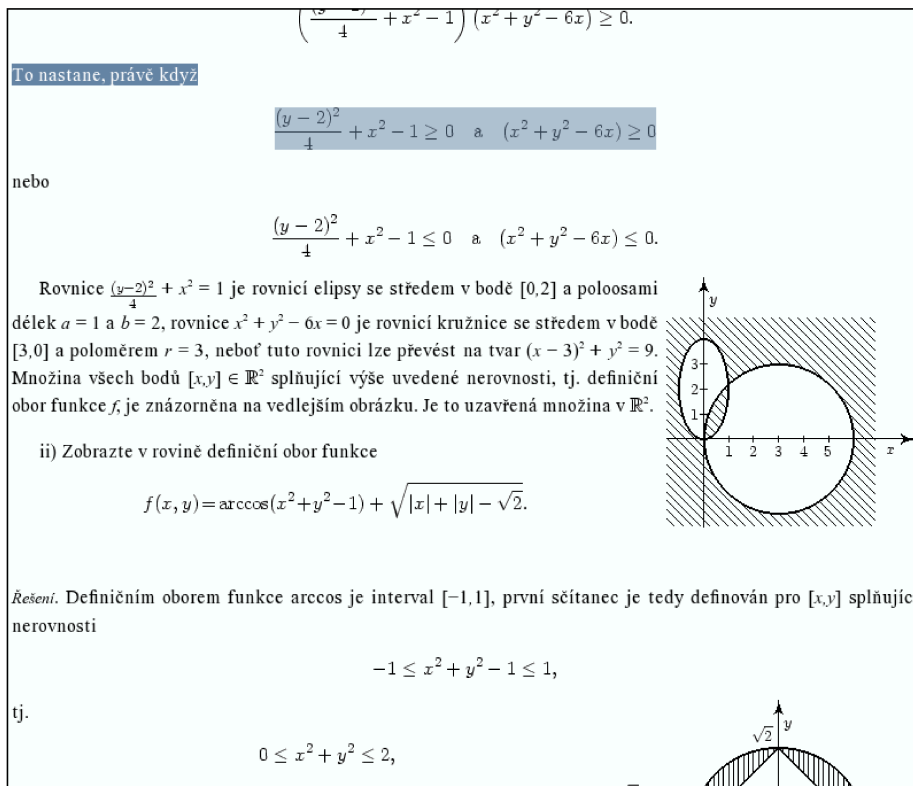


Figure 3: HTML output

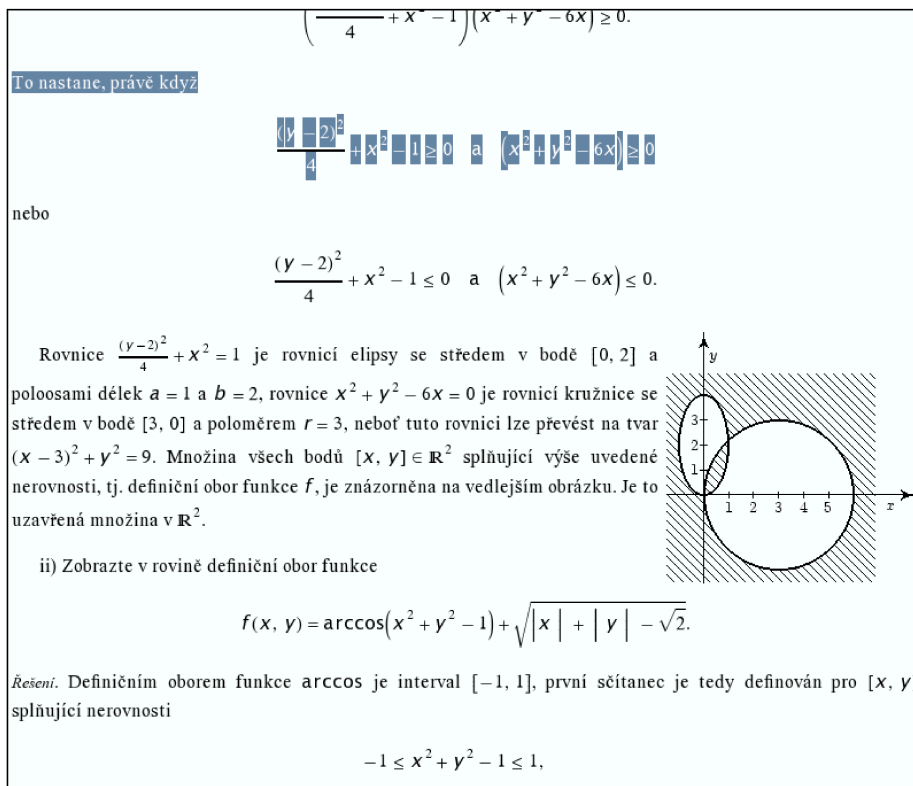


Figure 4: MathML output

5.2 XHTML + MathML

\TeX is very often used for typesetting of scientific texts that make heavy use of mathematical notation. A much more interesting alternative to HTML with math as images is the XML language MathML. The most complicated conversion process we tried was the conversion to XHTML + MathML output, especially for our case of highly mathematical texts.

There are still some complications for both the author and user when using MathML on the web. Firstly, there are different MathML implementations in web browsers, leading to different results. We achieved the best results with the \TeX 4ht `mozilla` compile option in combination with the Mozilla Firefox web browser (or other Gecko-based browsers). Secondly, the user has to have appropriate mathematical fonts installed. Information about necessary fonts, download links and installation instructions for Mozilla Firefox users are available on the Mozilla MathML Project web pages (<http://www.mozilla.org/projects/mathml/fonts/>).

A big advantage of \TeX 4ht is this MathML output possibility, which is very useful in case of mathematical texts — XHTML + MathML generation is similar to that for HTML:

```
htllatex filename.tex 'xhtml,mozilla'
```

When all goes well we get an XML file containing XHTML code that uses the MathML vocabulary for expressing mathematical formulae. You can see the result, fully scalable in a Mozilla Firefox window, in Figure 4 on page 122.

\TeX 4ht is very sensitive to having clean mathematical notation for MathML generation. For example, the expression

```
$M=\{x|x$ is odd $\}$
```

is correct \TeX code. But then \TeX 4ht is not able to pair the curly braces properly. One needs to use the right expression:

```
$M=\{x|x \mbox{ is odd}\}$
```

In complex situations, the math reformulation can be much more complicated or even impossible. In these rare cases \TeX 4ht offers an emergency solution — one can use pictorial object representation:

```
\ifweb
  \Picture*{}
\fi
$M=\{x|x$ is odd $\}$
\ifweb
  \EndPicture
\fi
```

This solution is not limited by the mathematical notation and may be used for any object with problematic rendering in MathML.

6 On-the-fly document generation

A second project where the single source approach is being used is the DML-CZ project [11, 1]. The project goal is not only to digitise a quarter million Czech and Slovak mathematical journal pages, but also to provide for parallel print and web-optimised generation for the digital-born data.

The digitised versions are being generated from scanned and preprocessed bi-tonal TIFF 600 DPI images. The following OCR task is being accomplished by a multistage process using FineReader (text) and InftyReader (mathematics) software [12]. InftyReader [13] is able to export the text in \LaTeX or MathML, together with positioning information. This allows typesetting of the OCRed \LaTeX or MathML (or both) data as text under an image: PDF \LaTeX supports this layered typesetting, e.g. using the standard `picture` environment. To make the text searchable the `cmap` package has to be used for texts in different languages.

An important factor is the size of the generated files — for print, resolution of printing device is usually demanded, while for on screen viewing lower resolution is sufficient. PDF allows different kinds of compression filters — since PDF 1.4, JBIG2 bi-level image compression is supported. PDF \TeX supports JBIG2 encoded figure inclusion in its recent version as well. In addition, Adam Langley has written the open source `jbig2enc` converter, as work supported by Google [8].

JBIG2 allows both lossless and lossy compression. For scanned content, the best approach is to get the best compression by accepting some loss of image data by using a symbol encoding where variation comes from printing errors [8]. JBIG2 compression ratio depends on context size — it gives better results when compressing page ranges instead of a single page only. Using slightly lossy compression and high context sizes, we are able to generate PDF images that have about 10–20 % of size of CCITT encoded and LZW compressed images. The parameters of `jbig2enc` allow for fine-tuned picture regeneration, even in an on-the-fly generation scenario.

Conclusion

Through examples of several projects, we argue that \TeX -based authoring and single-source publishing is a natural and effective way of preparing personalised documents for multiple output devices. \TeX 4ht is a very customisable tool for web publishing from \TeX sources, and the JBIG2 format for bi-tonal pictures saves space when generating many pictures (as in digitisation projects as DML-CZ or Google Scholar).

References

- [1] Miroslav Bartošek, Jiří Rákosník, Petr Sojka, and Martin Šárfy. Optical Character Recognition of Mathematical Texts in the DML-CZ Project, September 2006. accepted for publication as book chapter CMDE 2006 (A.K. Peters).
- [2] Zuzana Došlá and Ondřej Došlý. *Metric Spaces: Theory and Examples (in Czech)*. Masaryk University in Brno, second edition, 2000.
- [3] Zuzana Došlá, Roman Plch, and Petr Sojka. Matematická analýza s programem Maple: 1. Diferenciální počet funkcí více proměnných (Mathematical Analysis with Program Maple: 1. Differential Calculus). CD-ROM, <http://www.math.muni.cz/~plch/mapm/>, December 1999.
- [4] Zuzana Došlá, Roman Plch, and Petr Sojka. E-learning v matematice (E-learning in mathematics). DVD-ROM, December 2007.
- [5] Michel Goossens, Sebastian Rahtz, Ross Moore, and Bob Sutor. *The L^AT_EX Web Companion*. Addison-Wesley, Reading, MA, 1999.
- [6] Philip Babcock Gove and Merriam Webster. *Webster's Third New International Dictionary of the English language Unabridged*. Merriam-Webster Inc., Springfield, Massachusetts, U.S.A, January 2002.
- [7] Eitan M. Gurari. TeX4ht: L^AT_EX and T_EX for Hypertext. <http://www.cse.ohio-state.edu/~gurari/TeX4ht/>, February 2005.
- [8] Adam Langley and Dan S. Bloomberg. Google Books: Making the public domain universally accessible. *Proceedings of SPIE*, 6500:65000H, 2007.
- [9] Peter Meyer. Planning a single source publishing application for business documents, 2005. http://www.elkera.com/cms/articles/seminars_and_presentations/planning_a_single_source_publishing_application_for_business_documents/.
- [10] Peter Meyer. Introduction to single source publishing, 2006. http://www.elkera.com/cms/articles/technical_papers/introduction_to_single_source_publishing/.
- [11] Petr Sojka. From Scanned Image to Knowledge Sharing. In Klaus Tochtermann and Hermann Maurer, editors, *Proceedings of I-KNOW '05: Fifth International Conference on Knowledge Management*, pages 664–672, Graz, Austria, June 2005. Know-Center in coop. with Graz Uni, Joanneum Research and Springer Pub. Co.
- [12] Petr Sojka, Radovan Panák, and Tomáš Mudrák. Optical Character Recognition of Mathematical Texts in the DML-CZ Project, September 2006. accepted for publication as book chapter CMDE 2006 (A.K. Peters).
- [13] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INF_{TY} — An integrated OCR system for mathematical documents. In C. Vanoirbeek, C. Roisin, and E. Munson, editors, *Proceedings of ACM Symposium on Document Engineering 2003*, pages 95–104, Grenoble, France, 2003. ACM.
- [14] Edward R. Tufte. *Visual Explanations*. Graphics Press LLC, 1997.

Practical journal and proceedings publication on paper and on the web

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) cs dot bme dot hu
<http://www.inf.bme.hu/~pts/>

Abstract

Although \TeX is a reliable, high-quality and well-understood tool for authors writing their conference and journal articles, editors and typesetters face a much more difficult task when they want to compose articles for actual journal publication or preprints. We present organisational and software solutions for problems editors of journals and proceedings might face. As case studies we present issues and some conclusions regarding the production of the proceedings for two conferences we organised (Euro \TeX 2006 and the non- \TeX -related LME 2006 conference).

1 Introduction

We address problems during typesetting a collection of articles—usually a conference proceedings or a journal issue, from now on referred to as a “collection”. There are three parties cooperating: the authors, the editors and the printshop. Using our terms, an “editor” is someone who accepts articles from the authors, reviews articles, proofreads articles, typesets articles, or compiles a list of articles into a collection. We assume that editors work on \LaTeX article-like documents, and they convert any document they receive to this format. We also assume that the document class has already been designed by the typographer. We discuss converting articles to \LaTeX format, editing individual articles, and compiling a collection to be printed by the printshop, and also publishing it on the web as a set of PDF files.

We assume that time to be spent on editing is short, there are only a few editors, and not all the editors have a complete understanding of the whole publication process—some of them only review articles, others deal only with web pages, etc. We assume that there is a chief editor who would be able to do the whole job (except for peer review) if there was enough time.

We use two conferences we organised in 2006 as case studies. One of them is Euro \TeX 2006, an installment of the annual conference of the European \TeX community. Authors usually submit articles as \TeX source (most of them writing \LaTeX source using the document class the editors proposed), and the submitted material is of high typographic qual-

ity. That is, paragraphs, pages, tables and graphics look nice; graphics are in a scalable (vector) file format; extensive bibliographies arrive in a BIB \TeX format; and the layout is reasonably separated from the text so that editors can change the layout easily.

The other conference is LME GNU/Linux Conference 2006, known as LME 2006. It is one of the annual conferences of the Hungarian Linux and Unix community. Articles submitted are of varying linguistic and typographic quality. Most authors have never heard of \TeX ; many of them haven’t ever written an article before. They use plain text editors or OpenOffice (or an equivalent word processor) when writing documents. Editors have a lot of work to do with each article: file format conversion (from OpenOffice to \LaTeX), and proofreading and typesetting are slow. Some authors send graphics of extremely low quality or with unreadable captions—editors have to ask for a better version. They usually forget the bibliography or submit incomplete or incorrect entries—editors have to correct and supplement it.

We present the technology we found useful and best practices we have developed as a list of practical suggestions, some of them in imperative style. This is not meant to imply, however, that our solution is the only one feasible.

2 Organising work

Because of time pressure it is important that editors can work in a software environment most comfortable for them, and that they always have access to all the information they need. It is also important that document compilation works in a reasonably uniform way, so that e.g. line breaks don’t depend

on the computer the document was compiled on.

It is recommended that all work be done as soon as possible. For example, the mailing list and the repository can be created, mail client, chat client, repository client software and T_EX distributions and companion programs (like Ghostscript, OpenOffice and *sam2p*) can be installed way before the first article is submitted. The same applies to creating a document class (possibly from a layout designed by a typographer), collecting mail and chat addresses of each editor, providing access to the repository for the editors, and a little planning about the workflow.

2.1 The repository

The repository is a shared file store used by the editors. In a simplest case it is a shared folder on a server to which all editors have read and write access. However, using a version control system (such as Subversion [1]) is strongly recommended, because of these advantages:¹

- All past versions of files are available. If something goes wrong today, one can check out yesterday's state from the repository, and continue from there. We can also easily see what has changed, so there is a good starting point for finding what went wrong. Once the latest working version is identified, it is possible to revert to it easily.
- Each editor has their own (partial) copy of the repository. If the repository is lost in a server crash, editors can combine their copies and start a new repository. (This is quite inconvenient, but still a lot better than having to rewrite the whole collection from scratch.)
- Each write to the repository (called “commit”) is logged (who did it, when it happened, what files were affected and how). Thus if something goes wrong, we can find out who is capable of fixing the problem (usually the editor who introduced the problem is capable of fixing it, or he can provide the most relevant information for somebody else to fix it).
- If there are two different versions of the same file, there is no confusion as to which one is relevant (or more recent). The version control system automatically takes care of propagating changes in the right direction, without the need for manual review. This is a lot better than having several copies of the same file in a shared folder without knowing how they derived from

each other and which one is relevant for future work.

- Synchronising working copies is easy. If an editor makes a change, he executes a commit operation (which copies all local changes back to the repository), notifies others (usually on the mailing list) to update, and the other editors execute an update operation (which copies changes from the repository to their local copy). This works even if two or more editors are making changes on the same *text* file. If a conflict arises (i.e. two people made changes on the same line of a file), it has to be resolved by hand. Conflict resolution is distributed: the editor who was slower to make his change has to resolve the conflict. The chief editor is freed from the work of comparing different versions of the same file received in e-mail.
- If an editor uses several computers, a version control system provides seamless synchronisation between each local copy.
- Most version control systems provide a read-only web view. (We used `SVN::Web` for Subversion.) This is useful to allow the world to know the progress of the editing process. Authors or other organisers can be given access to the repository's web view, so that they can download recent and old versions of the files, they can view the differences with the file versions, and they can see a history of changes by examining the commit log.

Editors mustn't be allowed to share files in any other way than using the repository. The most common objection is that they haven't used such a system before, and there is no time to learn it now. However, if the chief editor writes a short tutorial about the version control system and the repository, and he helps other editors to install it (preferably via phone or voice chat), the learning time can be reduced to one or two hours. Using a version control system really pays off in both time and reliability. The size of the project being small isn't a valid argument against it either, because advantages are present even for single-file projects.

The chief editor must design the repository tree structure, and enforce it by moving files. It is not a problem that editors don't fully understand the structure, because with a good version control system (such as Subversion), files and folders can be moved and renamed easily. Some rules we used with EuroT_EX 2006: all filenames must be lower case English (with some additional restrictions on the allowed characters); file name length is not limited; all files received from the authors must be put

¹ The same advantages apply to software development — an area where version control systems have been used for decades with great success.

in the folder `art/00from_author/articlename`, all files needed by the article must be copied to `art/01recompiled/articlename`, and compiled there with only minimal modifications (in case of compilation problems, the author must be notified), and all compiled articles must be copied to and typeset in `art/02typeset/articlename` (with possibly a lot of modifications); the local texmf tree is in `texmf`, all necessary packages and fonts must be added there.

Everything possibly needed by editors should be added to the repository. This includes scripts, libraries, fonts and T_EX packages used, and also tutorials and guidelines. Software which is easy to install from packages (e.g. MiK_TE_X and Ghostscript) should not be added, however, but should be mentioned in a guideline along with the recommended version of each package. Files that can be regenerated (such as temporary files like `.aux` files and output files like `.dvi` and `.pdf`) shouldn't be added, except for milestone versions of output files (e.g. the `.ps` file sent to the printshop or the `.pdf` file sent to the proofreader).

Some version control systems distinguish between text and binary files. The difference must be understood, and files must be added in the proper mode. Both file types have advantages.

Editors should be encouraged to immediately correct each mistake they find in the repository. If they are not sure whether their correction is good, an easy solution is to ask them to contact the chief editor via chat, commit the change, and let the chief editor review it immediately. (The web view can be used to quickly get an overview on the changes.) The downside is that a wrong change might be in a repository for a few minutes. To avoid that, version control systems offer *branches*, but branches are usually too complicated to learn and use for newbies.

Sometimes editors forget to add a few files to the repository (for example, they add a nonstandard document class, but they forget to add the nonstandard packages loaded by the document class). This mistake can be prevented by asking the editors to have two working copies, and if they add a file in one working copy, they should recompile in the other one. Under Linux using *strace* is an alternative solution: running `strace -e open latex foo` prints all the files opened by *latex* when compiling *foo.tex*.

2.2 Mailing list

There should be a mailing list to which authors, editors and organisers can post; and editors and organisers can read the posted messages. (Multiple mailing lists can be created if a large traffic volume is expected.) Authors should be encouraged to

upload their articles to the web and post URLs to the mailing list. Alternatively, somebody should be made responsible for receiving articles from authors, adding them to the repository, and notifying editors about the article. It is generally a bad idea to receive articles on the mailing list, mostly because articles might be several dozen megabytes long.

The mailing list should be used only for notification and discussion, not for data transfer. All data to be worked on should be added to the repository, and others should be notified on the mailing list to update their working copy and do the appropriate action on the file. If there is a consistent proposal during a discussion, it also should be added to the repository instead of the mailing list.

2.3 Phone

Using the phone is the most efficient way that two distant parties can cooperate in real time. A phone call is extremely useful when one of the coworkers needs help (e.g. the commit resulted in a conflict, and the other party doesn't know how to resolve it), or when actions have to be synchronised (e.g. an editor commits a change he is not sure about, and the chief editor reviews it immediately).

When working on a computer connected to the Internet, one can make voice calls for free. Using Internet voice calls also gives the benefit of having free hands, so one can use his ears, eyes, mouth and fingers at the same time to solve a problem. Laptop users shouldn't rely on the built-in microphone of their laptop because of the terrible sound quality and the echo experienced on the other side of line. An external headset or a multimedia earphone (even as cheap as 5 euros) is a minimum.

2.4 Chat

Sometimes it might be feasible to use some chat (instant messaging) application instead of making a phone call. It is recommended that each editor have a chat account, and be online while working. However, we note that cooperation can be much more successful using the phone, because on the phone parties have each other's exclusive attention, with only a very few possible events to interrupt or suspend the conversation.

The chief editor should be registered in as many instant messaging networks as needed, and should use a multi-protocol client such as Gaim. Editors should use a client that beeps or pops up a window when a new message arrives, so they notice the message immediately. Web-based clients are thus out, because they don't notify the receiver.

2.5 Software

It is important to have software recommendations (including version numbers) for editors, so if the compilation output on two machines differs, it might be solved easily by switching to the recommended software.

On Unix we used $\text{te}\TeX$ 2 and 3 with some packages downloaded from CTAN to our local texmf tree. For \TeX source editing one could use any text editor; we recommended Kile and Kate. On Windows we used $\text{MiK}\TeX$ as a \TeX distribution and TeXnicCenter (and even Textpad) for editing.

We experienced font rendering problems and other bugs with Ghostscript 8.1x, so we recommended to upgrade to Ghostscript ≥ 8.53 .

As additional tools, we used the latest *sam2p* for raster image conversion, the latest *pdfconcat* for PDF concatenation and the *pdftops* tool from the Xpdf distribution for PDF to PostScript conversion.

We had our Subversion repository on a Unix server. For security, we allowed read-write access using `svn+ssh://` only. Users were authenticated using SSH public keys. We forced the `svnserve` command for these users in the `authorized_keys` file of SSH, with the parameters `-tunnel-user=...` `-t -r ...`. We also used an `authz-db` file in `svnserve.conf` to further tune access. `SVN:Web` was our read-only web frontend to Subversion. We patched it a little so that it could display commit log messages and files in a character set other than UTF-8. As a Windows client we recommended TortoiseSVN with PuTTY's *pageant* utility to avoid typing the passphrase for the public key again and again. We also prepared a tutorial on generating an SSH public key and setting up TortoiseSVN on Windows.

Our chief editor relied on the common scripting facilities of Unix (shell scripts, GNU Make and Perl), which helped his work a lot. However, other editors could work without those scripts if they wanted to, and they were in no way forced to understand the scripts. The recommended use of scripting was documented for them in a tutorial.

3 Tasks of the editors

Once papers start arriving, editors can start working on them. Although version control systems allow parallel modifications to the same file, this might result in conflicts, so we recommend that editors announce on the mailing list when they start or stop working on an article.

Usually one editor is able to typeset an article perfectly, except for proofreading, which should be done by as many people as possible. For LME

2006 each paper was reviewed by two experts, and checked for spelling and linguistic mistakes by two proofreaders, and we found errors even after that.

3.1 File format conversion

Recent versions of OpenOffice 2.0 contain a \LaTeX export filter, which can be used to convert word processor documents to \LaTeX . The filter handles paragraph breaks, bold and italic, emits simple Latin-1 and Latin-2 accented characters properly (without the *inputenc* package), and can export math formulas (we didn't test this feature thoroughly, because for LME 2006 we had only simple math formulas in documents). Since our documents had a lot of display verbatim material, and the export filter emitted it line-by-line, escaping each special character differently, we wrote a Perl script *postootex.pl* which post-processes the output of the export filter, that is, converts consecutive typewriter lines to a *verbatim* environment. The export filter was also quite loose on exporting font changes, it emitted superfluous `\rmfamily`, `\mdseries`, and font size change etc. commands even when there was no change at all. So we added code to the Perl script to remove these. We wanted a \LaTeX document that is easy to read and edit for humans, so we converted the markup input to Latin-2 (e.g. `\'a` to `á`). We also made the script remove the multitude of unnecessary braces inserted randomly by the export filter. Lists and enumerations were emitted almost properly, but the export filter insisted on reproducing the exact list formatting (margins, item width, etc.), so we removed this too, but with that we lost the list depths, so we had to check each nested list by hand. Exporting of tables, figures and floats was so preliminary that we decided to retype these elements by hand.

The usefulness of a custom Perl script to convert \TeX sources might sound questionable. We decided to write a script after frustration during the manual cleanup of the export filter's output on a 5-page document. We wrote the script so it tries to follow the \LaTeX syntax closely enough that it doesn't get confused e.g. by nested braces in an undelimited macro argument, and thus can clean up the source file reasonably well. The script didn't try to fix rare problems—its sole purpose was to save the editor the time of manually cleaning up the most common export glitches.

Raster images in OpenOffice documents didn't get converted (the `\includegraphics` command got exported, but it pointed to a nonexistent file). Fortunately, the OpenOffice document was a ZIP file, which contained the images as PNG and JPEG files,

which we could convert with *sam2p* to EPS and PDF. We didn't even try to export vector graphics, because authors sent such ugly figures that we decided to redraw them. We used Dia to redraw the figures, but we weren't satisfied with its formatting capabilities. It was a nightmare to change the visual appearance of the elements from the default.

None of the authors using OpenOffice supplied a structured bibliography, so we had to create the corresponding BIB_{TEX} source files by hand. The most tedious part of this task was to convert all URLs within the document to citations, and add fairly verbose entries to the bibliography database, looking up more information about the cited work on the Internet.

For Euro_{TEX} 2006, most authors followed the guidelines and used the L_ATEX document class we proposed, so no file format conversion was needed. Unfortunately, the final column width and font differed from those in the class we proposed earlier, so we got quite a number of overfull hboxes when recompiling articles. We also received articles in plain _{TEX} (!) and Con_{TEX}t, which we converted to L_ATEX by hand, heavily using the search and replace functionality in our text editor.

For Euro_{TEX} 2006 one of the authors sent a beautifully typeset article in PDF format, which we decided to include in the collection as is. Since the fonts and the column sizes were correct, we only had to add the running header and footer. We did this by importing the pages of the PDF file one-by-one as boxes with the *pdfpages* L_ATEX package.

3.2 Article compilation

We prepared shell scripts for Unix which set environment variables, run *mktexlsr* in the local texmf tree, and build _{TEX} formats with the necessary hyphenation patterns. This way it is easy to ensure that all editors work in the same environment. Should any difference arise (e.g. two editors have a different version of a L_ATEX package installed, and they get different output), it can be resolved by adding the file to the local texmf tree.

It is important that all documents be compilable automatically. If an editor manages to compile a document, he should immediately write a shell script to perform the compilation. E.g. if L_ATEX has to be run at most five times with a couple of BIB_{TEX} and *makeindex* runs in between, the shell script should contain the relevant commands in the proper order. It is not important to optimize for the number of L_ATEX runs — a possibly badly compiled document is a lot worse than a slowly but correctly compiled one. For clarity, another shell script should

be written that cleans up any temporary and output files. A Makefile can be used instead of shell scripts, but dependencies must not be indicated — a compilation should recompile the whole document from scratch. All scripts must share the same interface, so they can be called in a batch when the whole collection is recompiled, e.g. like this:

```
for DIR in *; do
  (cd "$DIR" && ./recompile.sh)
done
```

If the document contains raster images, they should be converted to both EPS and PDF, these files added to the repository, and the filename specified without extension in the parameter of `\includegraphics`. This way the document is compilable with both L_ATEX and pdfL_ATEX. We recommend *sam2p* for raster image conversion.

We decided that the recompilation of external graphics should not be part of the document compilation process. That is, when the document contains a figure drawn in Xfig, the Fig to EPS conversion isn't run when the document is compiled automatically. This gives us the advantage that even those editors can compile the document (and correct errors in the text) who don't have the appropriate graphics editors or converters installed. Asking them to install that extra software is not always feasible, because some graphics software needs specific operating systems or libraries.

All documents should be compiled to PDF with a fixed name (we used `compiled.pdf` for intermediate compilations and `final.pdf` for milestones). The reason why we are using PDF instead of PostScript is that PDF files are easier to manipulate (e.g. concatenate, add hooks) and they are also easier to preview in the web environment, so even visitors of the web view of our version control system can view milestones of typeset articles in PDF format. Using pdfL_ATEX is recommended (because it can break a line in the middle of a hyperlink, and it has some nice typographic add-ons), but if the document compiles with L_ATEX only, it should be converted to PostScript with *dvips*, and then converted to PDF.

Bitmap fonts must be avoided — a PDF with bitmap fonts looks ugly in Acrobat Reader, it is large and it renders slowly. Most _{TEX} fonts are available in vectorised (usually Type 1) format today (either in distributions or from CTAN). For example, the Bluesky fonts are the Type 1 outlines of Computer Modern (CM), the base _{TEX} font family. If the article uses the EC fonts, then the CM Super Type 1 outlines can be embedded to PDF, or the Latin Modern (LM) fonts can be used instead — but be aware

of the slightly different metrics and character shapes (such as the letter “ö”) between EC and LM.

Font installation and use can be cumbersome even if the font files are there in the proper folder of the local texmf tree. To avoid this problem, we used a custom Perl script *dff.pl* which wraps execution of all tools which embed fonts (currently *pdflatex*, *dvips* and *dvipdfm*) and provides the proper environment variables, command line arguments and font map files to these tools so that the right fonts will be found and used. The script also ensures that *pdflatex* and *dvips* use the same font map file.

The error and warning messages \LaTeX emits are useful, because they identify possible problems in the article. We decided to abort automatic compilation when a \LaTeX error is encountered, and thus force the editor to fix the error. We were quite permissive with warnings (including overfull box indications): we allowed compilation to continue, but wrote a Perl script which looks for warnings in the article log files, and we checked all these warnings after each milestone compilation. Finally we managed to get rid of all warnings. At some points we had to cheat, for example with long URLs in the bibliography it is quite hard to avoid the underfull hbox warning, so we just disabled this warning there by setting `\hbadness=10000`.

We used log analysis not only to find overfull boxes, but also badly embedded or missing fonts, and even articles accidentally omitted from the table of contents.

3.3 Editing

When reaching this point, the document is a valid \LaTeX article, with all its graphics converted to embeddable formats; the source markup is cleaned up enough for humans to edit; and there is a shell script that recompiles the article to PDF from scratch.

Simple editing is a straightforward task, which we took advantage of in LME 2006: we had a lot of volunteers for proofreading, so we quickly set up a tutorial for them on using the version control system, told them which files to start editing, and they could start contributing their changes.

We used standard tools for proofreading and typesetting corrections: the output-to-source navigation feature of the DVI previewer, and the big black `\overfullrule` to spot overfull boxes. For pages with complicated graphics or transformations, we previewed the PDF file instead of the DVI file. Xpdf was our preferred choice for PDF previewing, because it doesn’t have unnecessary GUI elements in its window, and it allows reloading the PDF file with a single keypress.

3.4 Concatenation

A collection is just a concatenation of the articles—except for the need for continuous page numbering to be maintained, a table of contents has to be generated, and there are some extra pages at the beginning and at the end.

We added the extra pages by introducing two special articles: `01Begin` and `99End`. The cover pages (two pages at the beginning and two others and the end) were part of these articles, but we had to strip these pages and send them separately to the printshop. Since we had to convert the document to PostScript anyway, the page range options to *pdf tops* solved the problem.

We didn’t generate the table of contents automatically; instead, we wrote a driver file which listed all the articles (with author, title and starting page number) in the order we wanted them to appear in the collection, and we typeset the driver file during the compilation of `01Begin`.

Automatic recompilation of the collection can work only if individual articles are already compiled automatically. We wrote a shell script which recompiled the whole collection. It also took care of propagating page numbers between articles. After each article compilation it counted the number of pages, modified the starting page number of the next article in both the driver file and in a helper file which would be `\input` by the document class. It took care of inserting an empty page so that each article began on an odd page. At the end it recompiled the two special articles in order to get the table of contents right. (No further compilation was necessary since we designed the TOC in such a way that the number of pages it occupied was constant.)

First we tried concatenating the PDFs using the *pdfconcat* tool. Unfortunately it doesn’t support PDF outlines (i.e. the table of contents tree) properly, so we switched to Ghostscript. Although Ghostscript 8.5x has support for concatenating outlines, the support had other glitches which prevented it from working with PDFs generated by \TeX . We prepared a small fix for that (which modified some of the PDF-writing operators such as *linkdest*), which also made hyperlinks work within the article. We also needed hyperlinks from the TOC to the article, but this was easy because we could use page-based links instead of symbolic ones, since we already knew the starting page number of the article.

We also tried the *pdfpages* \LaTeX package for concatenation, but this package didn’t support outlines or hyperlinks in source PDF files.

It was a key design principle in our workflow

to have automatic compilation. Since the work of the editors is judged based on the quality of the final output (both in print and the web), and humans tend to make mistakes (especially if they try to rush when the deadline is approaching), we wanted to have a document compilation policy which allows as few mistakes as possible in the final compilation. The more special cases the editors have to remember, and the more steps they do manually, the more mistakes they make. Automatic compilation minimizes these mistakes. It also increases the reliability of the editing process since if the computer of the chief editor gets broken during the editing, he can check out all the articles from the repository, and recompile the whole collection on any other computer with a single command. Unix shell scripts, Perl scripts and Makefiles helped us a lot for automating the compilation process.

3.5 Preparing for print

[3] gives a good technical introduction to the problems editors face when sending the work to the printshop, and it also gives several solutions for each problem (with both free and proprietary tools).

Printshops usually expect the text as colour-separated PostScript files. The cover pages have to be sent separately. The *psselect* tool can be used to select and reorder pages from a PostScript document, and options can be specified for *pdftops* to emit only a certain page range when converting from PDF to PostScript.

For high quality colour output one can use spot colours with the *xcolor* L^AT_EX package. As a simple alternative solution, one can create a PostScript document with colour, and later separate it. Separation means creating four copies of each PostScript page, each of these being grayscale, and the brightness values are used as C, M, Y and K components in the CMYK colour space. Aurora [2] is an old but working free tool which can do this conversion in pure PostScript. Using Aurora one processes the PostScript (or PDF) document four times, with settings for the individual component. Aurora wraps the *setgray*, *setrgbcolor*, etc. PostScript operators so that they will activate only one component of the specified colour. It also modifies the *image* and *colorimage* operators that draw raster images, but unfortunately it doesn't understand the image dictionary syntax introduced in PostScript Language Level 2. To overcome this, we implemented it in PostScript code which we load right after Aurora. Our code converts a PostScript image dictionary to a non-dictionary call of *image* or *colorimage*, and it also decodes indexed images manually.

The solution is quite slow (partly because of Aurora and partly because of our code); it processes a page with a colour image in about 10 seconds — but at least it is correct, because it hooks all affected operators at the proper place. Fortunately, we experience the slowdown only for raster images — colour text and vector graphics are rendered as quickly as without separation.

To make the job of the printshop easier, we prepared a script which separates the pages of a PostScript file to grayscale and non-grayscale. We took care of colour raster images manually, and we autodetected non-grayscale colours everywhere by looking at the colour-changing operators in the output of *pdftops*. Since this PostScript output has a quite simple syntax, we could find colour changes using regular expressions. Once the non-grayscale pages were found, we selected them with *psselect*, and renumbered the pages (back to the original) with a Perl script.

Printshops expect crop marks on each page. The *crop* L^AT_EX package can generate those marks. A few test pages should be sent to the printshop in advance so they can confirm that they get the crop marks where they expect them. In our case study projects we didn't use the *crop* package because it was more convenient for us to add the simple crop marks with a Perl script to the PostScript output of *pdftops*. The script also took care of enlarging the paper size. This way we could use our DVI and PDF previewers without having to see the enlarged page with the crop marks, and marks were added only to the PostScript file sent to the printshop.

3.6 Publishing on the web

We didn't want to have an HTML version of the articles, because converting L^AT_EX markup to high quality HTML is a difficult and time-consuming task which is hard to automate unless HTML export was in mind from the very beginning. We provided a HTML page with all the articles (with author, title, abstract and citations as BIB_TE_X source) and a PDF file for each article. We also provided a big, concatenated PDF.

PDF for the web differs from the printed document in:

- PDF for the web has outlines (structured table of contents) and in-document hyperlinks. All of these can be generated with the *hyperref* L^AT_EX package.
- PDF for the web has different margins, usually equal inner and outer side margins.
- PDF for the web doesn't contain crop marks.

- PDF for the web should contain scalable Type 1 fonts, because Acrobat Reader renders these fonts faster and nicer than bitmap fonts. Scalable fonts also reduce the file size.
- The size of PDF files for the web does matter. Raster images should be small. Fonts should be subsetted. Concatenated PDFs shouldn't contain the same font twice.
- PDF for the web can contain more pages with colour.

The first thing we did was add a “compilation mode” parameter to the compilation scripts. The document class also received this parameter, so it could generate slightly different output based on the mode (e.g. it could decide whether to load the *hyperref* package or not). With compilation modes we could also control if we need the overfull box indicator.

Ghostscript was smart enough to create a concatenated PDF with all fonts subsetted, except that pdfL^AT_EX had already subsetted fonts in the individual articles. So we turned font subsetting off in pdfL^AT_EX (changing all < signs to << in the font map file). This increased the size of intermediate PDFs substantially, but the final PDF became small.

We also wanted to have all fonts, including the base 14 fonts (like /Times-Roman) embedded, since we otherwise experienced accent positioning problems (e.g. with letter “ó”), since PDF viewers use different glyphs in standard fonts. To achieve this, we had to call Ghostscript with these parameters:
`-dCompatibilityLevel=1.3`
`-dPDFSETTINGS=/prepress`
`-dEmbedAllFonts=true`.

The sizes of raster images emitted by *sam2p* were small enough, but unfortunately Ghostscript insisted on recompressing the images (usually with suboptimal parameters). We solved this by writing the Perl script *pdfdelimg.pl* which extracted images from a PDF, and replaced them with dummy images. We run Ghostscript on these replaced PDFs, and we used *pdfdelimg.pl* again to replace images back in Ghostscript's PDF output. Our script distinguished dummy images by their dimensions.

In all other respects, Ghostscript produced small PDF output.

4 Conclusion

High quality text and math output is the most common reason why people like T_EX. Editors also appreciate the freedom they have when they design their workflow. They have several tools to choose from (many version control systems, many T_EX engines, many printer drivers, many converters), and they can customize the tools. Having the source

of the document in text files makes it possible to use a version control system for parallel file editing. Since there are multiple stages of compilation, there are multiple ways to hook in changes. Scripts can be written to automate compilation and generate both the printable and the web version from the same sources, with a single command. As far as we know, this set of features is unique to the T_EX editing workflow.

It is up to the chief editor precisely how to design the workflow and to what extent document compilation is automated. We tend to use a lot of custom scripts in our workflow, because we found that using scripts pays off in speed, quality of output and reliability, even when the script is run only once or twice; and we can also reuse our scripts in future projects. We admit that designing and setting up a good workflow needs quite a lot of software experience: the chief editor has to understand not only T_EX-, font- and PDF-related file formats and tools, but also version control systems (on both client and server side), web application installation, web page editing, mailing list management, and script programming. We believe that it is worth learning these and to improve the workflow gradually.

Communication between the editors is also important. The version control system ensures that editors have the relevant versions of all files they need, and also that they can make corrections to any file they want to. The mailing list and other communication channels can be used to distribute and synchronise work.

This article has presented some tools and techniques which can make collection preparation more productive and less painful. Since T_EX and its related tools are free software, there is a good chance that editors can find even better tools for their needs on the net. As tools and techniques continue to improve, working with T_EX becomes even more fun.

References

- [1] Ben Collins-Sussman, Brian W. Fitzpatrick, and Michael C. Pilato. *Version Control with Subversion*. O'Reilly, June 22 2004. <http://svnbook.red-bean.com/nightly/en/>.
- [2] T. Graham Freeman. Aurora: Colour separation with PostScript devices. Technical report, Australian Defence Force Academy, July 1994. <http://www.ctan.org/tex-archive/support/aurora/aurora.pdf>.
- [3] Siep Kroonenberg. T_EX and prepress. *TUGboat*, 25(2), 2004. <http://www.tug.org/TUGboat/Articles/tb25-2/tb81kroonenberg.pdf>.

An experimental CTAN upload process

Jim Hefferon

ftpmaint (at) tug dot ctan dot org

Abstract

Some experimental software may improve the way in which packages are handled at the Comprehensive T_EX Archive Network (CTAN).

1 Now

CTAN is run at three different sites, one in Germany, one in Britain, and one in the US. Any adding, deleting, or moving of files happens at one of these three. A custom program, written by Rainer Schöpf, ensures that a change at one is quickly reflected at the other two, within fifteen minutes. The more than one hundred other CTAN mirrors go at a different pace, usually syncing nightly.

New or updated material reaches us in three ways. Usually it sent by an author via a web form. Besides that, some authors send it via FTP, and some packages are automatically mirrored in from other sites. The author-sent cases could be either new packages or updates, while the automatic case only applies to updates. I will focus on the web uploads.

In the present system, a web upload triggers an email to the CTAN maintainers mailing list. The maintainer at the site receiving the material sees the email and handles the upload. This means unpacking the `.zip` or `.tar.gz` bundle in which the files were sent and examining the resulting files to check details such as license and placement. It may mean writing to the author or to the other maintainers, for instance to ask the author for documentation. After that, the maintainer runs Rainer's program to put the material into the archive and trigger the mirroring by the other core sites, and so ultimately by the additional mirrors.

Placing a package's files into the archive does not end its processing. Information about the package such as description and license — the package's metadata — needs to go into the *Catalogue*; this is done by Robin Fairbairns. Finally, distributions such as MiK_TE_X and T_EX Live repackage the material to meet the T_EX Directory Standard (TDS) and deliver it in this convenient form to typical end users.

The process above has some advantages. In particular, at an archive such as SourceForge where responsibility for how a package is offered lies with

the author, some percentage of the authors do not do a good job. But at CTAN the maintainers see that packages meet some standards. So a current strength of CTAN is that it is a wide-mouthed funnel, catching a range of submissions and narrowing them to a more uniform offering.

However, no doubt the process could be better. Here are a few concerns that we have heard.

1. Authors cannot conveniently edit the metadata.
2. There are delays of various kinds. One example is that package metadata often gets into the database only after the files are in the archive, so there is a period where the description does not match the package. Another example is that the web pages for the archive at <http://www.ctan.org/tex-archive> are usually regenerated nightly, so information about new materials is not current.
3. To be a core maintainer a person needs to run a server and there are people who could help with the archive but who oughtn't administer a system that is exposed to the Internet (including me).
4. The package gets installed by the maintainer whose site happened to receive the upload, so if that person is unavailable then there is a wait.
5. Many of the steps are done by hand, which can lead to errors.
6. At the time that a package is put in the archive and announced, it should be convenient for end users to install.

2 Developments

Users groups, notably Dante, have sponsored very helpful discussions of CTAN issues. In response, I have been working on software that is now being deployed and tested. The upload process described here still faces a fair number of hurdles. But some people have expressed interest and it is in an advanced enough state that the outline below may help these folks to get a rough understanding of what it does.

If you are not keen on CTAN internals then probably the feature that is the most interesting to you is also the most experimental. The T_EX Live team has a script to bring most packages from the CTAN tree over to the standard T_EX Directory Structure layout, that is, over to a format that could be dropped by an end user into their existing installation. The process described here wraps that script to make the TDS-ready material available as a $\langle package\ id \rangle.tds.zip$ bundle at the time that the package is put into the main CTAN archive. This is a regular ZIP file and users can `unzip` it right into their distribution tree, without much need for instructions. (This does not integrate with any package manager but it does allow users to easily place material that they want.)

To describe the process I will walk through the steps that a typical package would take to get from author to archive.

1. The author puts the package into a `.zip` or `.tar.gz` bundle. They visit CTAN's upload web address and first select whether the upload is a new package or an update of an existing package.

They then see the main upload form. Probably they fill out the simple version that asks only for name, license, and description. But more adventurous authors can get a form to specify more obscure attributes, such as the package home page.

If this is an update of an existing package then when the form appears it already has the metadata that is now in the database and the author just makes any changes. The author is asked separately for additional information such as any handling instructions (in the current system, the description and handling information goes in the same input box).

2. The system accepts the uploaded package and metadata. It places the metadata in the database, in a pool of not-yet-processed uploads. It sends an email to a list of people who can edit and install uploads, called here "editors".
3. The contributor's uploaded bundle is unpacked to a file tree by a program that runs periodically. (This does not happen as part of accepting the upload because the author's bundle must be unpacked in a secure way, in a chroot jail.)

This program does a few things beyond unpacking such as resolving text file line endings issues. When it finishes, it sends a notice to the email list of upload editors.

4. One of the editors sees the notification and logs into a web site listing the pool. They have a peek to see if the material is something that they could handle right now and if so then they claim responsibility for it.
5. This editor examines, possibly edits, and then approves the metadata left by the contributor. (Requiring that metadata be approved reassures authors that people they don't know cannot change the package's description.)

The editor can read, add, delete, or rename files. For instance, they can delete a `.svn` file that was accidentally included in the upload.

This page warns the editor if there are some problems. One example is that a warning will appear if the metadata says a `README` file exists but there is not one in the uploaded file tree. Another example is that a warning appears if an install will leave soft links dangling in the archive.

6. The author may have included in the upload their own $\langle package\ id \rangle.tds.zip$ file. If so, the editor can see its contents and compare with what T_EX Live now has for this package, if anything.

The editor can also push a button to make a new `.tds.zip` bundle, using T_EX Live's script. If the package is suitable for T_EX Live (which in most cases means only that it satisfies the license restrictions) then it can be placed in the local Subversion sandbox for later commitment to the T_EX Live repository. In either case, if the T_EX Live script does not succeed then the page makes that obvious.

7. The editor then pushes a button to install the material.

That puts the source files to the archive, say at `/macros/latex/contrib/⟨package id⟩`. Installation is done using the metadata so the database and the archive tree are consistent regarding the location, whether a `.zip` file exists of the directory contents, etc. Files are placed with Rainer's program, ensuring that these web-based installations are consistent with command-line installations.

The installation system also tends the database: it updates the metadata and the searchable documentation.

If there is a TDS bundle then the system puts it at a place related to where the source files went, such as `/install/macros/latex/contrib/⟨package id⟩.tds.zip`.

8. The installation routine sends an email to the editors list, telling people that the upload has

been handled, and for possible forwarding to the CTAN announcement mailing list.

9. If a `.tds.zip` bundle was queued in the local \TeX Live Subversion sandbox then the system will periodically try to commit the changes to the \TeX Live repository. One advantage of doing this at a separate time than the moment of installation is to guard against network connectivity problems between the CTAN site and the \TeX Live site. Another advantage is that when the \TeX Live folks are getting ready for a new release then this job can be shut off.

Material that comes in as an FTP upload goes through the same process, starting at step 3 (there is a way to associate metadata with the upload). This system has no way to handle materials that arrive automatically.

3 To do

Not every feature of the experimental system is described above; for instance, there is a way for authors to send changes to the metadata alone. And, because it is experimental, probably some of what is above will be changed if it ever reaches a production status. In particular, while the TDS feature appears promising, it is quite experimental.

So the upload process described here still faces a fair number of hurdles, both technical and non-technical. For one thing, where the current upload process is like a wide-mouthed funnel, the process described above has not been subject to any real-world testing for the same property. However, all the features described above exist, are now being developed and tested, and seem to solve at least some of the problems with the current package process.

4 Acknowledgement

Thank you to Karl Berry for help with the \TeX Live material.

TeX (Live) on Debian

Norbert Preining

Vienna University of Technology

Wiedner Hauptstraße 10

1040 Wien, Austria

preining (at) logic dot at

Abstract

TeX Live is a widely used TeX distribution incorporating most of the free (in the Debian sense) packages from CTAN, and binaries for many different architecture–operating system combinations.

Debian GNU/Linux is a popular operating system distribution based on the Linux kernel, containing only free [4] programs. Like most distributions of the Linux flavor, Debian has a strong package managing facility. Debian Etch was released in April 2007 with teTeX (version 3.0) and TeX Live (version 2005) packages. Future releases of Debian will contain only TeX Live packages due to the end of further development of teTeX.

This article describes the usage of TeX on Debian, from both a system administrator’s and a user’s point of view.

Thanks to Thomas Esser

To begin with, I want to take this opportunity to thank Thomas Esser for his incredible work on all TeX related things. His work has been the foundation of TeX Live and he himself continues to help and develop within the TeX Live distribution.

We all are very grateful to Thomas and wish him all the best with his future plans!

1 Rationale of Debian specific changes

As a big GNU/Linux distribution, Debian obliges package maintainers to prepare their packages in a standard way, requiring that (among other things):

- configuration files must be placed into the `/etc/texmf` hierarchy, and
- changes to configuration files are preserved during upgrade, but also preserved during a remove and reinstallation process.

(See the Debian policy document [5] for more details.) Most of the changes introduced in the Debian packages of TeX Live are due to the above two requirements. Other changes are due to the fact that many things (e.g., fonts, L^AT_EX-packages, programs) are already packaged for Debian and should be reused as far as possible.

The Debian TeX Task Force [3] has prepared a detailed document *Debian TeX policy* [2] and the more user oriented document *TeX on Debian* [1].

Finally, we want to stress that there is a certain overlap of Debian developers and TeX Live upstream maintainers, and the cooperation and bug forwarding/fixing has been mutually helpful.

2 Changing the configuration and file placement

2.1 Available TEXMF trees for users and system administrators

The following TEXMF trees are available. They are displayed below in the order they are searched, where earlier ones override later ones.

TEXMFCONFIG

Default location: `$HOME/.texmf-config/`
User-specific configuration files.

TEXMFVAR

Default location: `$HOME/.texmf-var/`
User-specific generated files.

TEXMFHOME

Default location: `$HOME/texmf/`
User-specific static input files, e.g., new L^AT_EX packages.

TEXMFSYSCONFIG

Default location: `/etc/texmf`
System-wide configuration files.

TEXMFSYSVAR

Default location: `/var/lib/texmf/`
System-wide generated files.

TEXMFLOCAL

Default location: `/usr/local/share/texmf/`
System-wide input files.

TEXMFMAIN

Default location: `/usr/share/texmf/`
System-wide, dpkg-managed input files (TeX add-on packages).

TEXMFDIST

Default location: `/usr/share/texmf-texlive`

System-wide, dpkg-managed input files (basic T_EX packages).

2.2 Configuration files

In the Debian Etch release, some configuration files are *not* shared between teT_EX and T_EX Live packages. The latter are in `/etc/texmf/texlive`, while the former are directly under `/etc/texmf`.

In the next release, with T_EX Live 2007 in Debian and teT_EX gone, all configuration files will be placed in `/etc/texmf`.

In any case, the `/etc/texmf` tree is by default the `TEXMFSYSCONFIG` tree, so any file placed in the proper location will override the respective file in `TEXMFMAIN`. This allows full control over the installation, but should be used with care only, as upgrades of the T_EX system will *not* attempt to merge changes in the shipped files into the replacement files you might put into `TEXMFSYSCONFIG`.

In addition to these files the packages ship some configuration files in `TEXMFSYSCONFIG`, and changes to these files will be preserved, and at upgrade time the system administrator informed about changes.

We will not list all the configuration files for teT_EX, T_EX Live 2005, and T_EX Live 2007, but instead pick the three most common situations occurring at normal usage: adapting the search paths and other `texmf.cnf` settings, upgrade or installation of a macro package (e.g., L^AT_EX style file), and installation and activation of a new font (family). We will only slightly touch the installation of new hyphenation patterns and formats.

3 Changing texmf.cnf

The central configuration file `/etc/texmf/texmf.cnf` is special, as it defines all search paths for (almost) all programs in the T_EX world. All the paths mentioned above are defined in it, but other behaviour (such as various size and security settings) is also controlled via this file.

Since many different packages can contribute to the final `texmf.cnf`, we adopted a method often used in Debian: We install separate parts of the configuration file into a special directory `/etc/texmf/texmf.d` and generate the final file from these snippets. Therefore, if a system administrator wants to change some setting, he should change the respective file in `/etc/texmf/texmf.d` and call `update-texmf`.

Take as an example the setting of `TEXMFHOME`: In `/etc/texmf/texmf.d/05TeXMF.cnf` one can find `TEXMFHOME = $HOME/texmf`. However, in my own institution's installation we had the input files always in `$HOME/texlib`, which I wanted to preserve.

So I change the given line in `/etc/texmf/texmf.d/05TeXMF.cnf` and call (as root) `update-texmf`.

The problem with this approach is that upon upgrade, either I have to reject changes of the file `05TeXMF.cnf`, or I have to change the settings after every change of `05TeXMF.cnf` in the Debian package. Here a bit of KPSE magic helps: As earlier settings in `texmf.cnf` override later ones, I can add a file `03local.cnf` to `/etc/texmf/texmf.d` and put the changed `TEXMFHOME` variable there.

Similar changes can be made for all the other settings in `texmf.cnf`.

If you really must change *as a user* some setting in `texmf.cnf`, you have to create your own `texmf.cnf` and override the `TEXMFCNF` variable.

4 Update/installation of a macro package, style file, etc.

This is a quite common task, as many packages are evolving very fast and sometimes newer versions are necessary. Let us go through the necessary steps for the `natbib` package. This procedure is the same for the T_EX systems on Debian and a 'default' T_EX Live installation.

4.1 Package update — system administrator

First you have to get all the files from your local CTAN node:

```
CTAN:/macros/latex/contrib/natbib
and put them into a temporary directory. After this
you run LATEX over all the .ins files to generate the
input files, and over all the .dtx files to generate the
documentation. You will end up with quite a number
of files; put the .sty files into $TEXMFLOCAL/
tex/latex/natbib, .bst files into $TEXMFLOCAL/
bibtex/bst/natbib, and if you wish the various
.dvi files (and any other documentation files) into
$TEXMFLOCAL/doc/latex/natbib.
```

After this, run `mktexlsr` and the next time any user of your system uses `natbib` the updated version will be used.

4.2 Package update — user

If you want to update `natbib` for yourself, and/or you don't have permission to change the `TEXMFLOCAL` directory, just replace it with `TEXMFHOME` and continue as above. As a normal user, calling `mktexlsr` is neither necessary nor desirable.

5 Installation and activation of a font package

Installation and activation of a font package is a bit more involved than just updating/installing a macro

package. We will go through this using the MathTimePro2 font set (available from Personal T_EX, Inc.).

5.1 Font update — system administrator

You should have received a zip file `mtp2fonts.zip`, which you should unzip into a temporary directory. MathTime is already shipped as a TEXMF-tree, so just copy all the files under `texmf` to the same location in `TEXMFLOCAL`, e.g.,

```
cp -ar texmf/* /usr/local/share/texmf
```

If you have some package `foo` that is not shipped as a TEXMF-tree, you have to install all the files you have obtained into the right places in `TEXMFLOCAL`, such as

```
.sty, .tex, .fd into $TEXMFLOCAL/tex/latex/foo
.map into $TEXMFLOCAL/fonts/map/dvips/foo
.tfm into $TEXMFLOCAL/fonts/tfm/comp/foo
.pfb into $TEXMFLOCAL/fonts/type1/comp/foo
.vf into $TEXMFLOCAL/fonts/vf/comp/foo
```

(Of course, some of these files may not be present.) After running `mktexlsr` again these fonts are now available to `tex`, but `dvips`, `pdftex`, `xdvi`, et al., will not yet recognize these fonts and will not display the fonts correctly.

For this you have to activate the respective `map` file which was (hopefully) shipped with the package. In our case there is the file `mtpro2.map` which we want to activate by default.

Here the Debian specific parts begin (but see below). The best way to do this is by:

1. adding a file `90local-mtpro2.cfg` into the directory `/etc/texmf/updmap.d`,
2. calling (as root) `update-updmap`, which generates the final `updmap.cfg` file from the snippets in `/etc/texmf/updmap.d`, and finally (as usual)
3. call `updmap-sys` to update the various configuration files for `dvips`, `xdvi`, etc.

Alternatively, you could put *all* your local adaptations into a file `90local.cfg`, if you prefer to keep them all together.

The above process describes the (native) Debian way to activate font maps. Due to the widespread recommendations on the web and user groups to activate a `map` file using a call like

```
updmap-sys --enable Map mtpro2.map
```

the version of `updmap(-sys)` in Debian has been adapted to *not* change the file `updmap.cfg` directly, but instead to enable and disable maps in `/etc/texmf/updmap.d/99local.cfg`. After this `update-updmap` is called, and then again `updmap-sys` for final operation.

Thus, changes made by `updmap-sys --enable` are not overwritten by a subsequent `update-updmap`.

Some reasons why Debian introduced the additional program `update-updmap` are:

- it does the job of the T_EX Live installer, which reads the information from the `tpm` files and activates the respective maps;
- several Debian packages can ship fonts and map files (e.g., `lmodern` or `cm-super`), and it must be possible for all of these parts to be activated and deactivated independently;
- the format of `updmap.cfg` cannot carry the necessary information on installation status and local changes (installed, removed, purged).

5.2 Font update — user

If a normal user without administrator rights wants to install and activate a new font set, he first has to install the fonts as described above, but instead of `TEXMFLOCAL`, he puts the files under `TEXMFHOME`.

When `update-updmap` is called by a normal user (`uid ≠ 0`) then it acts a bit differently: It merges all snippets present in `/etc/texmf/updmap.d/` and `~/.texmf-config/updmap.d/`, but if there are snippets with the same name, the one on the user directory shadows the system wide one.

Example Assume that a user has his own Sanskrit fonts, which provide fonts named `skt10`, etc., but the system file `10latex-sanskrit.cfg` already activates `skt.map`, which contains different definitions for these fonts. The following assumes the default for `TEXMFCONFIG`, namely `~/.texmf-config`.

To override the system-wide setting he would create a file with the same name, `10latex-sanskrit.cfg`, in `~/.texmf-config/update.d/` and call (as a user) `update-updmap`.

Thus, the files present on the system are as follows. In `/etc/texmf/updmap.d/`:

- `10texlive-base.cfg`
- `10texlive-latex-base.cfg`
- `10latex-sanskrit.cfg`

and in `~/.texmf-config/updmap.d/`:

- `10latex-sanskrit.cfg`.

With these settings the following files are used for *system-wide* `updmap.cfg` generation:

- `/etc/texmf/updmap.d/10texlive-base.cfg`
- `/etc/texmf/updmap.d/10texlive-latex-base.cfg`
- `/etc/texmf/updmap.d/10latex-sanskrit.cfg`

In contrast, the following files are used for *user-specific* `updmap.cfg` generation (the first two are the same):

- `/etc/texmf/updmap.d/10texlive-base.cfg`
- `/etc/texmf/updmap.d/10texlive-latex-base.cfg`
- `~/.texmf-config/updmap.d/10latex-sanskrit.cfg`

Finally the user must call `update-updmap`. This call will generate his own copy of `updmap.cfg` in `~/.texmf-var/web2c`. After this he can call `updmap` to generate the necessary configuration files for `dvips`, `xdvi`, etc., in `~/.texmf-var`.

Note that changes in `/etc/texmf` are *not* automatically carried over to the user files. So in case something is going wrong the user should again call `update-updmap` and `updmap`.

6 Hyphenation patterns and formats

To install new hyphenation patterns and new formats you can follow the above example concerning fonts, with `update-language` and `update-fmtutil` taking the place of `update-updmap`, the path components `language.d` and `fmt.d` the place of `updmap.d`, and `fmtutil(-sys)` the place of `updmap(-sys)`.

7 Backports for Debian Etch

The Debian T_EX Task Force is also trying to provide backports of all the necessary packages for Debian Etch (stable). Currently we are able to provide binaries for the i386, AMD-64, and PowerPC architectures. All that is necessary is to put the following three lines (sorry for the editorial line breaks necessary here) into the `/etc/apt/sources.list` file:

```
deb http://people.debian.org/~preining/TeX/tl2007/
deb http://people.debian.org/~preining/TeX/context/
deb http://people.debian.org/~preining/TeX/lmodern/
```

All packages shipped on these pages are signed with my Debian GPG key available in the Debian keyring or various key servers.

8 Further developments

Things are evolving very fast at the moment. While Debian Etch ships with T_EX Live 2005, the 2007 release of T_EX Live is already present in Debian Sid and testing ('lenny'), bringing X_YT_EX to the Debian world.

At the same time we provide independent packaging of ConT_EXt and LuaT_EX to make Debian the ideal play ground for further developments.

People interested in cooperation are invited to contact our mailing list [3], take a look at the Subversion repository [6] where all the packaging scripts are available, not only for T_EX Live, but also Latin Modern, ConT_EXt, LuaT_EX, cm-super, etc., or contact me directly.

References

- [1] T_EX on Debian. <http://people.debian.org/~preining/TeX/TeX-on-Debian/>.
- [2] Debian T_EX policy. <http://people.debian.org/~frank/Debian-TeX-Policy/>.
- [3] Debian T_EX Task Force mailing list. <http://lists.debian.org/mailman/listinfo/debian-tex-maint>.
- [4] Debian Free Software Guidelines contained in the Debian Social Contract. http://www.debian.org/social_contract.
- [5] Debian Policy. <http://www.debian.org/doc/debian-policy/>.
- [6] Subversion repository of the Debian T_EX Task Force. <http://svn.debian.org/wsvn/debian-tex>.

Epspdf: Easy conversion between PostScript and PDF

Siep Kroonenberg*

Rijksuniversiteit Groningen
Department of Economics
Groningen, the Netherlands
siepo at cybercomm dot nl

Abstract

This article introduces epspdf, a converter between eps, PostScript and pdf, which can be run either via a graphical interface or from the command-line.

Introduction

When preparing a L^AT_EX document, it is often convenient to have graphics available both in eps and in pdf format. Epspdf² improves on previous solutions by having both a CLI (command-line interface) and a GUI, by converting both ways, using pdftops from the xpdf suite,³ and by various new options, which were made possible by round-tripping between PostScript and pdf.

Sample applications

Case 1: Converting a PowerPoint slide to pdf and eps A.U. Thor writes a paper in L^AT_EX and creates his illustrations with PowerPoint. He likes to turn these into pdf graphics, so that he can process his paper with pdflatex.

From PowerPoint, he ‘prints’ to an eps file (see the appendix). The Windows Print dialog is rather insistent on giving the eps file an extension ‘.prn’. He loads the graphic in epspdfk (see figure 1), where the .prn file is accurately identified as eps. He checks the ‘Compute tight boundingbox’ option, selects pdf output format, and clicks ‘Convert and save’. Some annoying black boxes flit across his screen, but soon a message ‘Conversion completed’ appears. He presses the ‘View’ button and Adobe Reader displays what he hoped to see.

He might as well replace the eps with one with a good boundingbox, so he converts the pdf back to eps. It may save epspdf some work if he first unchecks the boundingbox checkbox.

When viewing the resulting eps with GSview, there is once more a lot of whitespace around the figure. Hunting around in the GSview menus, he finds ‘EPS Clip’ and ‘Show Bounding Box’ in the

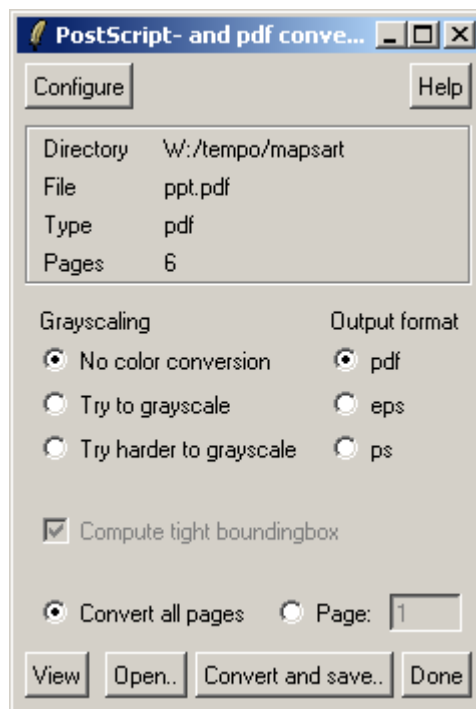


Figure 1: Main window of epspdfk (MS Windows)

Options menu, and with either option checked, he can see that all is well.

Case 2: Converting multiple slides from a PowerPoint presentation to pdf graphics A.U. Thor, encouraged by his previous success, adds several new graphics to his PowerPoint file. Since epspdfk can handle multi-page documents, he prints the entire document to a PostScript file, which he loads in epspdfk.

He notices that the box with file info doesn’t tell him the number of pages. For general PostScript files, there is no sure-fire quick way to get this information.

* This article has previously appeared in MAPS 34.

² <http://tex.aanhet.net/epspdf/>, also on CTAN.

³ <http://www.foolabs.com/xpdf/>

He checks ‘Convert all pages’ and sets ‘Output format’ to pdf. After conversion, the info box now gives him the number of pages.

He writes the first page to a pdf file with a tight boundingbox, reloads the complete pdf, then writes the second page to a pdf, and then wonders whether the command-line might not be more convenient.

He reads the epspdf webpage and manual, browses the epspdf directory and comes up with a batchfile `epspdf.bat` containing the following line:

```
"C:\Program Files\epspdf\bin\ruby.exe"
"C:\Program Files\epspdf\app\epspdf.rb"
%1 %2 %3 %4 %5 %6 %7 %8 %9
```

(everything on one line) and types

```
epspdf -b -p 3 ppt_slides.pdf fig3.pdf
```

Then he outdoes himself in cleverness and does the remainder with one command (everything on one line):

```
for %f in (4 5 6) do
  epspdf -b -p %f ppt_slides.pdf fig%f.pdf
```

Case 3: Creating cropped typeset samples

Mrs. TeX-HeX writes a paper for MAPS about her adventures with L^AT_EX. She wants to display typeset examples with her own fonts and formatting, not with those of MAPS. So she creates a L^AT_EX document containing one sample per page, and makes sure, with a preamble command `\pagestyle{empty}`, that each sample is the only content on its page. She compiles the document to a pdf and extracts the samples with a tight boundingbox, in the same way as in the previous example.

Case 4: Embedding fonts into an existing pdf

Ed Itor is collecting papers in pdf format for a conference proceedings. The printshop tells him that one of the submitted pdf files doesn’t have all its fonts embedded, which is a no-no, even though the font is just Courier and Ed Itor doesn’t quite understand the fuss.

Luckily, when converting PostScript to pdf, Ghostscript can embed standard PostScript fonts (Times etc.) even if they are missing in the PostScript file. Ed Itor goes to the Configure screen and verifies that ‘Intended use’ is set to ‘prepress’. With this setting, converting to PostScript and back to pdf does the trick.

Warning: It is quite possible that the original document was created with slightly different versions of the missing fonts than the ones included by Ghostscript. Usually this will cause no problems, but one might want to check the result anyway.

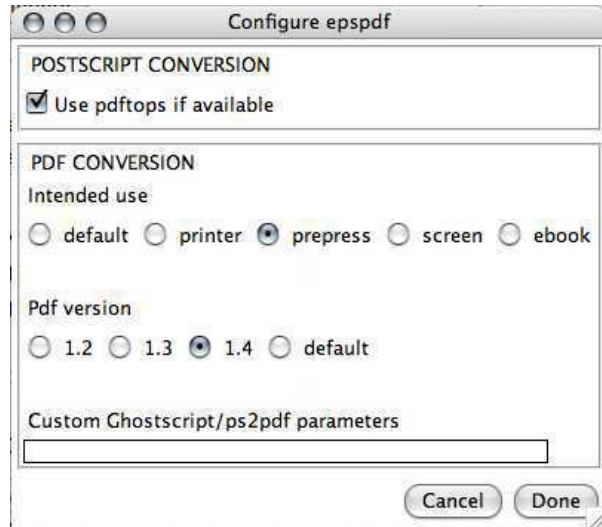


Figure 2: Configuration screen (Mac OS X)

Some implementation details

The program is written in Ruby and Ruby/Tk, and uses Ghostscript and pdftops from the xpdf suite for the real work. The installation instructions in the download and on the web page explain how to obtain these programs.

The program consists of several modules. The main ones are a main library `epspdf.rb` which does double duty as the command-line program, and a Ruby/Tk GUI main program. Conversions are organized as a series of single-step conversion methods and an any-to-any conversion method which strings together whatever single-step methods are required to get from A to B.

I have included all conversions and options into the program that easily fit into this scheme.

Configuration Epspdf saves some conversion options between sessions. Under Windows it uses the registry, under Unix/Linux/Mac OS X a file named `.epspdfrc` in the user’s home directory. Besides some precooked options, advanced users can also enter custom options for Ghostscript (GUI and CLI) and for pdftops (CLI only).

The Windows setup program For Windows, `epspdf` is available as a Windows setup program. This includes a Ruby/Tk subset, so there is no need for a full Ruby and Tcl/Tk install. This Ruby/Tk subset is adapted from one generated with `RubyScript2Exe`.⁴

But Windows users can also manually install from a zipfile if they already have Ruby and Tcl/Tk.

⁴ <http://rubyforge.org/projects/rubyscript2exe/>

Mac OS X Under Mac OS X, epspdf mostly duplicates functionality from the Preview application. However, when faced with problem files it is nice to have an alternative converter (see figure 2). From the epspdf homepage you can download a double-clickable and dockable AppleScript applet for starting epspdftk.

Appendix: exporting PostScript from Windows programs

Using a printerdriver Often, the only way to get eps or PostScript from a Windows program is by ‘printing’ to a PostScript file.

For this, you need to have a PostScript printer driver installed. You can pick ‘FILE’ for printer port. A suitable driver which comes with Windows is Generic / MS Publisher Imagesetter.

Pay attention to printer settings: in the Print dialog, click ‘Properties’, then ‘Advanced’ (on either tab). In the ‘Advanced Document Settings’ tree, navigate to first ‘Document Options’ then ‘PostScript Options’. (See figure 3.)

For ‘PostScript Output Option’ the default setting is ‘Optimize for speed’. Change that to ‘Optimize for Portability’ or ‘Archive Format’, or, for single pages only, ‘Encapsulated PostScript’. These non-default options presumably produce cleaner PostScript code, without printer-specific hacks. Experiment with this and other options if you run into problems (e.g. bad-looking screen output, or part of a graphic getting cut off, or conversion to bitmap).

What works best may depend on your Windows version: under Windows 2000, Archive worked best for me, but Taco Hoekwater warns me that this option was unusable in older Windows versions.

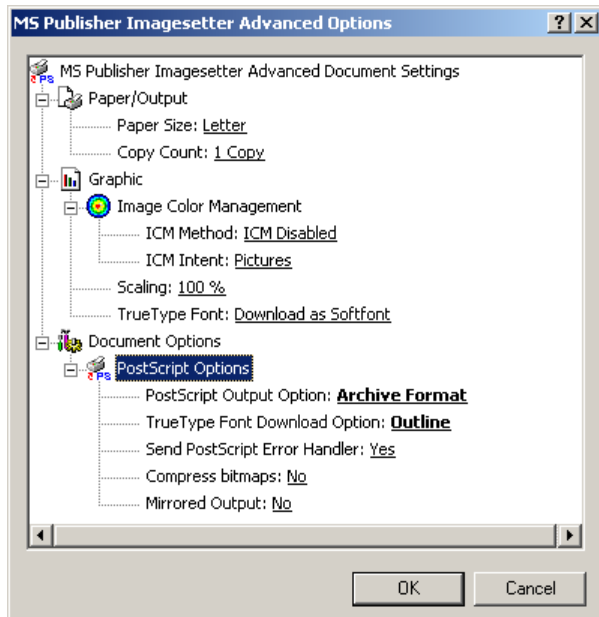


Figure 3: Configuring the Windows PostScript driver

Another setting here to pay attention to is ‘TrueType Font Downloading Option’. Pick ‘Outline’, not ‘Automatic’ or ‘Bitmap’.

Using a program Other possibilities for generating PostScript or pdf are the TpX and wmf2eps programs, which both can read Windows wmf and emf files and also have options to write clipboard contents to an emf file. Wmf2eps uses its own virtual PostScript printer driver in the background. For faithful conversion, pick wmf2eps; for subsequent editing, choose TpX. Both programs are available from CTAN.⁵

⁵ <http://www.tug.org/ctan.html>

pdfTeX 1.40: What's new

Martin Schröder
Crüsemannallee 3
28213 Bremen
Germany
martin@pdftex.org
<http://www.pdftex.org>

Abstract

The latest version of pdfTeX, 1.40, was released at the start of 2007. We will present its new features and have a look towards the future.

1 The past

pdfTeX turned 10 on 15th March (Hàn Thê Thành renamed T_EX2PDF to pdfTeX on the 15th March 1997), and we want to present the latest release. But let us first look back at our previous release, which was pdfTeX 1.30.0 on 1st August 2005. Since then there have been six intermediate releases fixing bugs (mainly security problems with XPDF); version 1.30.6 was released on 16th February 2006. Its main enhancements were improvements in the handling of PNGs (alpha channel and transparency, 16-bit colour and gamma correction), macros for timekeeping, random numbers, string conversions, and file functions. Also pdfxTeX was gone; all enhancements were now in pdfTeX and pdf ϵ -TeX.

2 The present — 1.40

After 17 months of development we released pdfTeX 1.40.0 on 1st January 2007. At the time of writing there have been five intermediate releases fixing bugs; version 1.40.5 was released on 31st July 2007. pdfTeX 1.40.x is included in T_EX Live 2007 and MiKTeX 2.6. The main internal change is that we merged all the sources (i. e. change files) for T_EX, ϵ -TeX and pdfTeX into the two files `pdftex.web` and `pdftex.ch` (for Web2C etc.). This makes maintaining the sources much simpler. Also pdf ϵ -TeX is gone as a separate program; pdfTeX now contains ϵ -TeX.

2.1 PDF

pdfTeX is now able to generate object streams, a feature of PDF since PDF 1.5 [1]. A PDF file consists of objects and a cross-reference table for fast access



This work is licensed under the Creative Commons Attribution-Non Derivative Works 2.0 Germany License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/2.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

to these objects. But formerly only the stream part of the objects could be compressed. So if a PDF had many non-stream objects, its size could not be reduced efficiently. Object streams are a kind of meta-objects; they can contain many (up to 256) non-stream objects. From pdfTeX 1.40 onward object streams can be compressed as a whole, which leads to smaller PDF files. The generation of object streams is controlled by the `\pdfobjcompresslevel` parameter (only with PDF ≥ 1.5):

- 0 The default; no object streams are generated.
- 1 Object streams are generated, but the Document Information Dictionary and included PDFs are not compressed.
- 2 The Document Information Dictionary is not compressed, everything else is.
- 3 Everything is compressed.

Another improvement leading to smaller PDFs is that pdfTeX now writes out the widths of the characters in the fonts (`/Widths`) with a higher precision and so rarely has to position characters separately. Previously this was done with a lower precision, leading to many adjustments of single character positions in the PDF.

The new primitive `\pdfastlink` now gives the object number of the last link created with `\pdfstartlink`.

PDF/X, the new ISO standard of PDF, requires the setting of `/ModDate` and `/Trapped` in the Document Information Dictionary. These keys have now default values that can be overridden with `\pdfinfo`.

Also the additional PDF statistics in the log file about the number of objects etc. are now correct; previously they were written out too early and thus missed the objects for e. g. embedded fonts.

2.2 JBIG2

pdfTeX can now handle another format of image files: JBIG2. JBIG2 is an image compression standard

for binary images, developed by the Joint Bilevel Image Experts Group (which is also responsible for the JPEG standard). It is suitable for both lossless and lossy compression. According to a press release from the Group, in its lossless mode JBIG2 typically generates files from 3 to 5 times smaller than Fax Group 4. PDF supports JBIG2 since PDF 1.4, but previously there were no free encoding programs for JBIG2, only decoding programs, so the OSS world was unable to generate JBIG2 files. This changed in 2006 when Google sponsored the development of a free encoding program (`jbig2enc`). pdfTeX (and L^AT_EX with a recent `pdftex.def`) support JBIG2 files with `.jbig2` or `.jb2` suffixes.

2.3 Colour stacks

Colour in pdfL^AT_EX has an old problem: If you have different colours on the page and in the footnotes, you'll probably get the wrong colour after a page break. The `pdfcolmk` package tried to fix this, but it was a kludge. To fix this, pdfTeX 1.40 introduces support for colour stacks; L^AT_EX gets this through `pdftex.def` and some packages (e.g. `pdfcolfoot`). Colour stacks are handled with these commands:

- `\pdfcolorstackinit [page]` initializes a colour stack and expands to the number of the stack. With `page` you get a stack that is reset at the start of every page.
- `\pdfcolorstack <stack number> push {<colour>}` saves the `<colour>` on the stack and outputs it.
- `\pdfcolorstack <stack number> pop` removes the topmost colour from the stack and sets the now topmost value.
- `\pdfcolorstack <stack number> current` gets the topmost colour from the stack and sets it, but doesn't change the stack.
- `\pdfcolorstack <stack number> set {<colour>}` sets the topmost colour of the stack, but doesn't change the rest of the stack.

2.4 Transformation matrices

PDF (like PostScript) uses transformation matrices for positioning objects. Before pdfTeX 1.40, matrix changes were done and hidden inside `\pdfliteral` nodes, but pdfTeX doesn't parse the argument of `\pdfliteral` and so does not know the new settings of the transform matrix, which might conflict with pdfTeX's use of the matrix. pdfTeX 1.40 adds new primitives to save pdfTeX from parsing `\pdfliteral`'s argument and to notify pdfTeX about matrix changes for use in calculating link and anchor positions.

- `\pdfsetmatrix{<a> <c> <d>}` is the equivalent of `\pdfliteral{<a> <c> <d> 0 0 cm}`
- `\pdfsave` is the equivalent of `\pdfliteral{q}`
- `\pdfrestore` is the equivalent of `\pdfliteral{Q}`

Some remarks:

- T_EX already supports translations, thus the matrix is limited to four values, for scaling and rotating.
- There are some restrictions about `\pdfsave` and `\pdfrestore`:
 - They must be properly nested.
 - A pair must start and end in the same group at the same level.
 - A pair must start and end at the same position.

Happily these restrictions are satisfied by the `graphics` package.

The latest `pdftex.def` uses these primitives.

2.5 General enhancements

- pdfTeX now offers limited support for namespaces:
 - `\pdfprimitive<TEX-primitive>` executes the original `<TEX-primitive>`, even if its definition has changed. Thus

```
\let\relax\undefined
\pdfprimitive\relax
```

 works and doesn't give an error.
 - `\ifpdfprimitive<TEX-primitive>` is true if `<TEX-primitive>` still has its original meaning.
- `\ifpdfabsnum` and `\ifpdfabsdim` are like `\ifnum` and `\ifdim`, but don't care about signs.
- The memory areas for PDF objects (`obj_tab`) and names (`dest_names`) now grow dynamically as needed, making corresponding settings in `texmf.cnf` obsolete.
- `\pdfsavepos` now also works in DVI mode.
- The resolution of PK files is now read from `texmf.cnf` if it hasn't already been set in the format or document.
- In almost all cases of fatal pdfTeX errors (i. e. if the resulting PDF would have been invalid anyway) no PDF is generated.
- The format of warnings and error messages has been revised and unified.

- If called with `-version` pdfTeX now tells the versions of the libraries compiled with and actually used:

```
Compiled with libpng 1.2.15; using libpng 1.2.15
Compiled with zlib 1.2.3; using zlib 1.2.3
Compiled with xpdf version 3.01
```

- pdfTeX now can be switched into a draft mode with `-draftmode` and `\pdfdraftmode=1`. In draft mode pdfTeX does everything it normally does, but does not write a PDF and does not read the contents of included images, thus speeding up the execution. This is useful e.g. if you know you need another two L^AT_EX runs to get the references right.

2.6 Fonts and HZ

- pdfTeX now supports subfonts: All needed map entries are generated automatically together with the Unicode mappings.
- pdfTeX can generate ToUnicode entries for Type1 fonts with `\pdfgentounicode` and `\pdfglyphtounicode`.
- Previously with font expansion in autoexpand mode for every expansion a complete new font was included in the PDF. Now the font is only included once and gets expanded on the fly by using the text matrix. This leads to smaller PDFs and enables the use of HZ with TrueType fonts and even non-embedded fonts (e.g. Times-Roman).
- Hàn Thế Thành describes more improvements in his paper [2].

2.7 Shell escape

If the first character of a file name for `\openin`, `\openout` and `\input` is a “|” (and `\write18` has been enabled), the rest of the file name is executed as a command. Some examples:

```
\openin1= "|ls -l"
\loop \unless \ifeof1
  \read1 to \cs \message{\meaning\cs}
\repeat
```

outputs the filenames in the current directory.

```
\openout1= "|sort >alphabet.tex"
\write1{b}
\write1{a}
\write1{c}
\closeout1
```

generates a sorted file.

The shell escape feature is available in all Web2C-based T_EX engines, e.g. X_YT_EX and pdfT_EX.

3 The future

The future of pdfT_EX is luaT_EX: The pdfT_EX team will take over the maintenance and development of luaT_EX once its initial development has been finished. This will offer support for Unicode and OpenType and integrate Lua [3], thus finally giving T_EX a proper programming language. pdfT_EX will still be maintained for those needing a time-proven engine, but new features will be added only to luaT_EX.

References

- [1] Adobe Systems Incorporated. *PDF Reference*. Sixth edition, 2006.
- [2] Hàn Thế Thành. Font-specific issues in pdfT_EX. In *Proceedings of the EuroT_EX 2007 conference*, 2007.
- [3] R. Ierusalimschy, L. H. de Figueiredo, W. Celes. *Lua 5.1 Reference Manual*. 2006.

X_ƎTeX Live

Jonathan Kew

SIL International

Horsleys Green

High Wycombe

Bucks HP14 3XL, England

jonathan_kew (at) sil dot org

1 X_ƎTeX in TeX Live

The release of TeX Live 2007 marked a milestone for the X_ƎTeX project, as the first major TeX distribution to include X_ƎTeX (version 0.996) as an integral part. Prior to this, X_ƎTeX was a tool that could be added to a TeX setup, but version and configuration differences meant that it was difficult to ensure smooth integration in all cases, and it was only available for users who specifically chose to seek it out and install it. (One exception to this is the MacTeX package, which has included X_ƎTeX for the past year or so, but this was just one distribution on one platform.) Integration in TeX Live, in contrast, provides near-universal availability and a more standardized configuration, which should simplify setup, use and support.

Special thanks to Karl Berry for his encouragement and support through this process, and to all the TeX Live builders and testers on various platforms who helped to make this possible.

1.1 Key features

The two most significant features of X_ƎTeX as found in TeX Live remain the same as they have been since its first appearance: support for the use of the host operating system's fonts (PostScript, TrueType, or OpenType) with no TeX-specific setup, and including layout features defined in the fonts; and extensive support for Unicode, including complex Asian and other scripts. With this release, users on all platforms have the option of using the same OpenType fonts in TeX documents as in mainstream GUI applications, including access to all the rich typographic features found in modern fonts.

As an example of the simplicity X_ƎTeX brings to font usage, consider the present article. This is written using the `lƎtugproc` class. Running this in X_ƎLaTeX, the lines:

```
\usepackage{fontspec}
\setmainfont[Mapping=tex-text]
                {Adobe Garamond Pro}
\setmonofont[Scale=MatchLowercase]
                {Andale Mono WT J}
```

Note: This article is based on the author's presentations at both the EuroBachTeX 2007 and TUG 2007 conferences, but is printed in a single *Proceedings* issue to avoid duplication.

in the preamble are sufficient to set the typefaces throughout the document. These fonts were installed by simply dropping the `.otf` or `.ttf` files in the computer's Fonts folder; no `.tfm`, `.fd`, `.sty`, `.map`, or other TeX-related files had to be created or installed.

Release 0.996 of X_ƎTeX also provides some enhancements over earlier, pre-TeX Live versions. In particular, there are new primitives for low-level access to glyph information (useful during font development and testing); some preliminary support for the use of OpenType math fonts (such as the Cambria Math font shipped with MS Office 2007); and a variety of bug fixes.

1.2 Hyphenation setup

A long-standing problem with integrating X_ƎTeX has been the variety of hyphenation patterns for various languages, which are written using a variety of character encodings and various ways to represent those encodings in 7-bit or 8-bit files. Because X_ƎTeX interprets 8-bit text files as Unicode (UTF-8) by default, many old hyphenation files cannot be read as-is. This in turn meant that the X_ƎLaTeX format could fail to build, depending on the user's language configuration.

Older releases of X_ƎTeX made some attempt to address this by including modified versions of some of the hyphenation files from TeX, adapted to load correctly as Unicode patterns. However, ensuring that these were installed in the right place for X_ƎTeX to find them (without affecting other engines or replacing standard files) was problematic.

In TeX Live 2007, this situation has been addressed by modifying the `language.dat` file so that hyphenation files are loaded via "wrappers" (except for those that are simple ASCII files, which are already Unicode-compatible). The wrapper files, provided in TeX Live in `texmf-dist/tex/generic/xu-hyphen`, test whether the format is being built by X_ƎTeX, and if so they redefine the input encoding and/or `\catcodes`, active character definitions, etc., so that the patterns will be loaded as Unicode data. Figure 1 shows an example of such a wrapper file; in this case, the German vowels with umlauts and the β character need Unicode-compliant definitions, in place of those found in the original hyphenation file. The precise details vary, of course, depending on the structure and encoding

```

% xu-dehyphn.tex
% Wrapper for XeTeX to read dehyphn.tex
% Jonathan Kew, 2006-08-17
% Public domain
\begingroup
\expandafter\ifx\csname XeTeXrevision\endcsname\relax
\else
  \catcode`\?=7
  % Define the accent macro " in such a way that it
  % expands to single letters in Unicode
  \catcode`\`=13
  \def"#1{\ifx#1a??e4\else \ifx#1o??f6\else \ifx#1u??fc\else
    \errmessage{Hyphenation pattern file corrupted!}%
    \fi\fi\fi}
  % - patterns with umlauts are ok
  \def\n#1{#1}
  % - define \3 to be character "00DF (\ss in Unicode)
  \def\3{??df}
  % - define \9 to throw an error
  \def\9{\errmessage{Hyphenation pattern file corrupted!}}
  % - duplicated patterns to support font encoding OT1 are not wanted
  \def\c#1{}
  %
  \let\PATTERNS=\patterns
  \def\patterns{% at the \patterns command in dehyphn.tex...
    \endgroup % end group containing local definitions from dehyphn
    \begingroup % and start our own (to match \endgroup in dehyphn)
    \PATTERNS % and then load the real patterns
  }
\fi
\input dehyphn.tex
\endgroup
\endinput

```

Figure 1: xu-dehyphn.tex, a typical hyphenation wrapper file from the TeX Live setup

of the pattern file being loaded, but similar techniques can generally be used.

In the longer term, reorganization and standardization of the hyphenation files, perhaps co-ordinated with work in OpenOffice.org (which uses a very similar hyphenation algorithm) would be a useful project. However, this will require not only a good understanding of the language and encoding issues, but also interaction with license holders or maintainers of all the existing patterns. Meanwhile, the current setup with xu- wrappers has proved to be a workable interim solution.

1.3 Package configuration

Another common problem for XeTeX users in the past has been that some popular L^ATeX packages (e.g., graphics, color, geometry, crop, hyperref, and others) depend on knowing the intended output driver (direct PDF generation with pdfTeX, dvips, dvipdfm, etc.) in order to use the correct implementation-specific methods to control the output. Many such packages attempt to detect the TeX engine in use and automatically choose the appropriate driver. However, with XeTeX being a new engine, existing packages were unaware of it.

This situation is improving, as some major packages have added a test for XeTeX and now choose the appropriate driver options. For others, including important cases like geometry and crop, TL2007 includes configuration files in the xelatex subtree that provide the proper setup. In most cases, therefore, users should find that the packages work transparently in XeTeX just as with other TeX engines and drivers.

One important package that did not work transparently with XeTeX in the TL2007 release is pgf; however, since the release in February, pgf has also been updated so that it recognizes the XeTeX engine automatically.

1.4 The ArabXeTeX package

A new package by François Charette provides an ArabTeX-like interface for typesetting languages in Arabic script, using standard Unicode-based fonts. As shown in figure 2, this supports both literal Unicode input of Arabic text, and ArabTeX transliterations, and can work with any OpenType font, including complex calligraphic styles such as Nastaliq script. This package was created after the current TeX Live release, but can be obtained from CTAN and works with the existing XeTeX version.


```

% set up Cambria Math for roman, symbol and extension families
\font\1="Cambria Math:script=math" at 10pt
\font\2="Cambria Math:script=math;+ssty=0" at 7pt
\font\3="Cambria Math:script=math;+ssty=1" at 5pt
\textfont0=\1 \scriptfont0=\2 \scriptscriptfont0=\3
\textfont2=\1 \scriptfont2=\2 \scriptscriptfont2=\3
\textfont3=\1 \scriptfont3=\2 \scriptscriptfont3=\3

% use Cambria Math with italic mapping for family 1
\font\1="Cambria Math:script=math;mapping=math-italic" at 10pt
\font\2="Cambria Math:script=math;mapping=math-italic;+ssty=0" at 7pt
\font\3="Cambria Math:script=math;mapping=math-italic;+ssty=1" at 5pt
\textfont1=\1 \scriptfont1=\2 \scriptscriptfont1=\3

% set mathcodes (many are predefined in xetex.fmt)
\XeTeXmathcode\^-="2 "2 "2212 % minus sign
\XeTeXmathcode\Σ="1 "2 \Σ % summation

% some control sequences...
\XeTeXmathchardef\sum="1 "2 \Σ \XeTeXmathchardef\prod="1 "2 \Π
\XeTeXmathchardef\intop="1 "2 \int \XeTeXmathchardef\infty="1 "2 \∞
\XeTeXmathchardef\geq="3 "2 \≥ \XeTeXmathchardef\leq="3 "2 \≤
\XeTeXmathchardef\pi="7 "1 \π

% using Unicode characters in math
$$ f(x) = a_0 + \sum_{n=1}^{\infty} \left( a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right) $$

```

Result, using an OpenType math font:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right)$$

Figure 4: Defining and using Unicode math characters

When using a complete OpenType math font such as Cambria Math, it may be necessary to load the font several times with different character mappings and OpenType features.

2.3 Inter-character token insertion

A new feature in X_ƎTeX version 0.997 is the ability to insert arbitrary token lists in between normal text characters, without complex macro programming. This is designed primarily to support requirements of Japanese and Chinese typography, where special spacing controls are needed in certain cases such as between ideographs and adjacent punctuation characters.

To support this feature, each character has a “class” known as `\XeTeXcharclass`, a bit like an extra `\catcode`, but ignored by normal TeX operations. But whenever two printable text characters occur next to each other, X_ƎTeX will check their class values, and if a token list has been defined for this class pair it will be inserted between the characters. Such a token list may contain arbitrary TeX material, although the most useful possibilities are probably various forms of `\skip` and `\penalty` (to control spacing and breaking), and font changes (making it possible to

automatically switch fonts for different scripts within Unicode text, without requiring embedded markup).

For example, the default `xetex` and `xelatex` formats initialize most `\XeTeXcharclass` values to zero, but assign all the CJK ideographs to class 1. We can take advantage of this to allow Chinese characters to be included in running text without additional markup, even though the default body font does not support them; a simple example is shown in figure 5. While this technique is not a universal substitute for proper language and font markup in the source document, it can greatly simplify the author’s task in some mixed-script situations.

2.4 Graphite font support

The initial version of X_ƎTeX, on Mac OS X only, supported special font features such as contextual swashes, ligatures, alternate glyphs, etc., by means of Apple’s AAT font technology. Later, support for OpenType font features was added, based on the ICU layout library; this enabled X_ƎTeX to provide complex font support across multiple platforms.

A third font layout technology, designed to support the requirements of non-Latin scripts, minority languages, and scripts not yet in Unicode, is SIL’s Graphite system (<http://scripts.sil.org/RenderingGraphite>).

Conventional scoping of registers — An experiment in $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$

Gerd Neugebauer

In Lerchelsbühl 5

64521 Groß-Gerau (Germany)

gene (at) gerd-neugebauer dot de

www.gerd-neugebauer.de

Abstract

$\text{T}_{\text{E}}\text{X}$ provides groups as a means to restrict the visibility of registers. This construction is well known in the $\text{T}_{\text{E}}\text{X}$ world but does not coincide with groups as known from other programming languages. If we refrain from storing the register value in a global array we can come to the alternate solution of storing it in the control sequence used to access it. With this variant we can provide a means to define an arbitrary number of registers which follow the same scoping rules as variables in Pascal-like languages.

$\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ is a reimplement of $\text{T}_{\text{E}}\text{X}$ in Java. It is developed with extensibility and configurability in mind. The idea of an alternative storage for registers can be implemented in $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ as an extension. We show which steps are required for such an implementation. In this manner the extensibility of $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ is demonstrated.

1 Registers and scoping

`plain.tex` provides macros to handle the allocation of registers. For this document we want to restrict our considerations to count registers. The other register types can be handled analogously. Here the macro `\newcount` can be used to allocate a new count register:

```
\newcount\abc
{\abc = 42
 \showthe\abc
}
```

In $\text{T}_{\text{E}}\text{X}$ any changes to registers are recorded. When the group closes, the old values are restored. Thus any changes to registers in a group are automatically local. This can be overwritten with the keyword `\global`.

Let us have a look at the same construction in another programming language. As an example we use Java. The same considerations hold for many languages of the Pascal family.

```
{ int abc = 42;
  System.out.println(abc);
}
```

The grouping reduces the scoping of the variable `abc`. It is defined within the group and not visible outside. If a variable with the same name is defined before the group then this variable is hidden by the new definition in the group.

The explicit declaration of the local variable in Java arranges things so that the new variable is activated and any previous declaration is hidden.

2 Storage in $\text{T}_{\text{E}}\text{X}$

Coming back to $\text{T}_{\text{E}}\text{X}$ an alternative interpretation comes to mind. Whenever a register is modified in a group then an automatic declaration of a new variable is introduced and initialized.

One way to come closer to conventional programming languages with $\text{T}_{\text{E}}\text{X}$ would be to introduce typed variables following the conventional rules for scoping and initializing.

$\text{T}_{\text{E}}\text{X}$ stores the values of registers in $\text{T}_{\text{E}}\text{X}$ memory. This memory is written to the format file when a `\dump` is performed. Besides the register values, (macro) code is stored in $\text{T}_{\text{E}}\text{X}$ memory.

All we need is a primitive which behaves like a count register but stores the value somewhere else — making it accessible via the primitive only.

3 $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$

The $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ project (\rightarrow <http://www.extex.org>) has the aim to produce a reimplement of $\text{T}_{\text{E}}\text{X}$. The implementation language for this reimplement is Java. The major design decisions put modularity and configurability into the forefront.

As one consequence $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ is assembled out of many components. Those components provide defined interfaces. This makes it simple to write replacements for existing components and provide new

components to extend the system. This extensibility makes it easy to experiment to some extent with new ideas. In the following sections we will see one example of such an experiment.

$\epsilon\lambda\text{T}\text{E}\text{X}$ is currently under development. Even though large pieces are in place, $\epsilon\lambda\text{T}\text{E}\text{X}$ is not yet ready for production. Any help to get things finished is very welcome. If you are interested in participating in $\epsilon\lambda\text{T}\text{E}\text{X}$ development, contact the developers on the developer list, which can be found via the $\epsilon\lambda\text{T}\text{E}\text{X}$ web site.

4 Writing a new primitive for $\epsilon\lambda\text{T}\text{E}\text{X}$

According to our considerations we want to have a new primitive which behaves like a count register but stores the value within the code and not in the context. In addition we need a primitive `\integer` to dynamically create such integers. Then we can write the following TEX code:

```
{\integer \abc = 42
 \showthe\abc
}
```

First we start with implementing the code for the count-equivalent. This code needs to have several properties to behave like a count register:

- It needs to assign a new value when executed. This means that

```
\abc=123
```

works if `\abc` has the meaning of the new primitive.

- It needs to act as an assignment; this means that `\afterassignment` has to be taken into account. This means its token is expanded after the assignment has taken place.
- It needs to be advanceable. This means that the following works:

```
\advance\abc by 123
```

- It needs to be multiplyable. This means that the following works:

```
\multiply\abc by 123
```

- It needs to be divideable. This means that the following works:

```
\divide\abc by 123
```

- It needs to provide the count value upon request. This means that the following works:

```
\count0=\abc
```

- It needs to provide value for primitives `\the` and `\showthe`. This means that the following works:

```
\showthe\abc
```

- It needs to expand to the tokens making up its value.

5 Providing a definition

To start with we create a new class. This class lives in a package named `extex.tutorial`. In addition we use a bunch of imports from $\epsilon\lambda\text{T}\text{E}\text{X}$. Since the imports are usually filled in by the IDE, we omit them (like the comments which are assumed to be filled in by the reader).¹

```
package extex.tutorial;

import org.extex.core.count.Count;
// a bunch more imports omitted
```

Next we declare the class. It is derived from an abstract base class which takes care of the assignment. Each of the properties we want to have is declared with the help of an interface. `Advanceable` describes that the primitive can be used after the primitive `\advance`, `Divideable` describes that the primitive can be used after the primitive `\divide` and so on. Each of these interfaces contains a single method which needs to be implemented.

```
public class IntPrimitive
  extends
    AbstractAssignment
  implements
    Advanceable,
    Divideable,
    Multiplyable,
    CountConvertible,
    Theable,
    ExpandableCode {
```

Since we want to store a count value with the code we first create a private field. The data type `Count` encapsulates a count value. It has the methods to access and manipulate it. In its core it contains a `long` value to store a number in.

```
private Count value = new Count(0);
```

But before we come to implement the interfaces we have to define a constructor. The constructor

¹ To be honest, the exact package structure of $\epsilon\lambda\text{T}\text{E}\text{X}$ is subject to some changes until the final version 1.0 is released.

takes one argument — the name of the primitive — and passes it to the constructor of the super-class.

```
public IntPrimitive(String name) {
    super(name);
}
```

Now we can start with the first method `assign`. It takes four parameters with the following interfaces:

`Flags` contains the indicators for the prefix arguments like `\global`. The primitive can consume the flags and react differently upon their values. Since our primitive does not use prefixes this argument is simply ignored.

`Context` contains the equivalent to the $\text{T}_{\text{E}}\text{X}$ memory — anything contributing to the state of the interpreter is stored here. The `Context` is also stored in a format when `\dump` is invoked.

`TokenSource` provides access to the scanner and the parsing routines. It can be used to acquire further tokens or even higher order entities.

`Typesetter` contains the typesetter of the system. The typesetter produces nodes which might be stored in boxes and finally sent to the backend. The primitive can send characters or instructions to the typesetter or simply request some information from it.

We will see these parameters again with each of our methods.

```
public void assign(Flags prefix,
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    source.getOptionalEquals(context);
    Count newValue = CountParser.parse(
        context, source, typesetter);
    value.set(newValue);
}
```

The implementation first consumes an optional equal sign and then parses a following count value. Finally we can set the internal count to this new value.

Assume that we have assigned the new primitive to the control sequence `\abc` — we will see the details later. Then we can do the following:

```
\abc = 1234
```

This simply assigns a new value to the variable. But we have also used the infrastructure of an assignment. Thus the tokens stored in the token

register `\afterassignment` are inserted after the assignment:

```
\afterassignment=\x
\abc = 1234
\y
```

Right now we can assign a new value to the variable. Since we want to see what we have done, we implement the method `the` which converts the value back into tokens to be used by the primitives `\the` and `\showthe`.

```
public Tokens the(Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException,
    CatcodeException,
    ConfigurationException {

    return context.getTokenFactory().
        toTokens(value);
}
```

The main task of creating a list of tokens is provided by a token factory. This is an application of the factory pattern. The factory is attached to the context and can be retrieved from it.

Next we have to take care of `\advance`. In $\epsilon\chi\text{T}_{\text{E}}\text{X}$ the implementation of `\advance` decouples the operation from the implementation of the primitive. Thus it is possible to add further primitives which can be used after `\advance`. This goal is reached with the help of the interface `Advanceable`. When the token has the meaning of code which implements this interface then the control is passed to the methods defined in the interface to carry out the operation. We use this feature to make our primitive applicable to `\advance`.

The method uses the parsing routines in $\epsilon\chi\text{T}_{\text{E}}\text{X}$ to acquire the optional keyword by and the value for a count register. This value is added to the variable stored in this primitive.

```
public void advance(Flags prefix,
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    source.getKeyword(context, "by");
    Count by = CountParser.parse(
        context,
        source,
        typesetter);
    value.add(by);
}
```

The same technique used for `\advance` is used for `\divide` as well. Thus we just have to implement the associated interface `Divideable` and provide the following method:

```
public void divide(Flags prefix,
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    source.getKeyword(context, "by");
    Count by = CountParser.parse(
        context,
        source,
        typesetter);
    value.divide(by);
}
```

And once again the same trick for `\multiply`: We implement the interface `Multiplyable` and provide the following method:

```
public void multiply(Flags prefix,
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    source.getKeyword(context, "by");
    Count by = CountParser.parse(
        context,
        source,
        typesetter);
    value.multiply(by);
}
```

Converting into a count value is expressed with the interface `Countconvertible` which has one method `convertCount`. This method delivers the count value as `long`. Since we have the variable in our private field we can just take the value from there.

```
public long convertCount(
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    return value.getValue();
}
```

Finally we provide a means to use the primitive in an expandable context. When tokens are expanded — in contrast to executed — we simply push the tokens representing the value to the token source. Thus they are read and processed afterwards.

```
public void expand(Flags prefix,
    Context context,
    TokenSource source,
    Typesetter typesetter)
    throws InterpreterException {

    try {
        source.push(
            context.getTokenFactory().
                toTokens(value));
    } catch (CatcodeException e) {
        throw new InterpreterException(e)
    }
}
```

The method is slightly complicated by the handling of an exception which might come from the creation of the tokens. This exception is simply remapped and passed upwards.

This is all we need to do to implement the new primitive.

```
}
```

6 Putting things into place for testing

Now we are finished writing our new primitive as a Java class. But how can we make use of it? First of all we have to compile it with a Java compiler and put it into a jar — say, `abc.jar`. $\epsilon\lambda\TeX$ is installed in a directory. This installation directory contains a subdirectory named `lib`. All jars contained in this directory are automatically considered when classes are loaded. Thus we put `abc.jar` into this directory.

Next we make use of a quick extension mechanism to try out our fine new primitive. Later we will use the configuration mechanism of $\epsilon\lambda\TeX$ for this purpose. But now we simply use the dynamic extension mechanism which allows us to bind some Java code to a primitive. To do so we need to load the unit `jx`. Units in $\epsilon\lambda\TeX$ are collections of primitives. For instance there is a unit `tex` containing the `\TeX` primitives.

One of the primitives contained in $\epsilon\lambda\TeX$ — i.e. in the unit `extex` — is the primitive `\ensureloaded`. It takes one argument in braces which is the name of a unit and loads this unit if has not yet been loaded into the interpreter.

This primitive is used now to load the unit `jx`:

```
\ensureloaded{jx}
```

After the unit `jx` has been loaded we can make use of the primitive `\javadoc` provided by this unit. This primitive is similar to the primitive `\def`. It takes a control sequence and a list of tokens enclosed

in braces. The control sequence gets a new meaning. This meaning is determined by the Java class named in the tokens argument:

```
\javadef\abc{extex.tutorial.IntPrimitive}
```

Now we can use the primitive `\abc` as shown above. This is enough for testing. Nevertheless it is discouraged since it uses an implementation specific extension. The recommended way is to use the configuration facility described later.

7 Defining new variables

The definition of each new variable with `\javadef` is a little bit clumsy. Our original plan was to define any new variable with `\integer`. It takes a control sequence and the initial value. This can be accomplished with a small definition of the following kind:

```
\def\integer#1{%
  \javadef#1{extex.tutorial.IntPrimitive}%
  #1}
```

This approach works but has the disadvantage that the resulting macro does not interact properly with `\afterassignment`. The primitive `\javadef` is an assignment. Thus the `afterassignment` token would be inserted just after the definition but before the initial value has been read.

To overcome this problem and gain some more insight into the definition of primitives in $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ we implement this primitive in Java as well.

The class itself is started as shown before. Since the task is much simpler we do not need to declare a lot of implemented interfaces.

```
package extex.tutorial;

// a bunch of imports omitted

public class IntDef
  extends AbstractAssignment {
```

The constructor propagates the name to the super class — as before.

```
public IntDef(String name) {
  super(name);
}
```

Finally we have to implement the `assign` method. Here we can make use of the `TokenSource` to acquire a control sequence. Now we create a new instance of our class `IntPrimitive`. The argument is the name of the variable. This name is extracted from the control sequence token.

Now we can use the method `assign` of this new instance to assign the initial value. Finally we bind

the new instance to the control sequence token. This binding makes use of an optional prefix argument `\global`. The prefix is read and cleared in one step. The clearing avoids an error message about unused prefix arguments.

The `\global` prefix allows us to define a global variable — even within a group. This extension was not on our initial agenda, but is easily implemented.

```
public void assign(Flags prefix,
  Context context,
  TokenSource source,
  Typesetter typesetter)
  throws InterpreterException {

  CodeToken cs =
    source.getControlSequence(
      context,
      typesetter);
  IntPrimitive code =
    new IntPrimitive(cs.toString());
  code.assign(Flags.NONE,
    context,
    source,
    typesetter);
  context.setCode(cs,
    code,
    prefix.clearGlobal());
}
```

Now we are finished and can use the primitive.

```
}
```

We have postponed the configuration of $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ until we have the primitive. Now we can fill this omission.

8 Configuring $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$

The encouraged way of extending $\epsilon\mathcal{X}\text{T}_{\text{E}}\text{X}$ is by configuring a new unit. The configuration of a unit is an XML file following a particular schema. The outer tag is `unit`. It can have attributes. The mandatory attribute we are using is the attribute `name` which is used to specify the name.

As an inner tag we are using `primitives`. Inside this tag all additional primitives of this unit are listed with `define` specifications. The defines need attributes. The attribute `name` specifies the name of the control sequence to assign the definition to. The attribute `class` specifies the Java class. This class needs to implement the interface `Code`. This class is instantiated and bound to the control sequence.

The configuration file `tutorial.xml` has the following contents:

```
<unit name="tutorial">
  <primitives>
```

```

<define name="integer"
      class="extex.tutorial.IntDef"/>
</primitives>
</unit>

```

We have placed the compiled Java files in a jar. The configuration file `tutorial.xml` has to be placed in the same jar file. To be found, it has to be placed in a certain package. This is the package `config.unit`. Now we can load it like we have done with the unit `jx`:

```
\ensureloaded{tutorial}
```

9 Aliasing variables

With the variables introduced here we can use `\let` to create aliases for a variable. `\let` creates a new binding for a control sequence to the same code as an existing control sequence. With our implementation in mind it is immediately apparent that a modification of one variable at the same time also modifies all aliased variants. This is illustrated in the following example:

```

\integer\x=42
\let\y=\x
\x=123
\showthe\y

```

In this code `\x` and `\y` share the same content. After assigning 123 to `\x` this value also shows up when printing `\y`.

This trick can be used to access a variable which is hidden by a local variable. In this case you can make an alias before defining the local variable:

```

\integer\x=42
% . . .
{\let\y=\x
 \integer\x=123
 \showthe\y
}

```

10 Variables and name spaces

In [1], namespace support for $\epsilon\chi\text{TEX}$ was presented. Namespaces primarily act on primitives. This collides with the access to registers via one primitive — for instance `\count` for all count registers. The allocation macro `\newcount` from `plain` can be used to assign a control sequence to a certain count register. This control sequence is subject to the name space visibility. Nevertheless the control sequence can be bypassed.

With the variables introduced in this paper we can overcome this deficiency. The variables introduced interact in a natural way with the namespace concept of $\epsilon\chi\text{TEX}$.

11 Conclusion

We have seen an alternate way of defining variables in $\epsilon\chi\text{TEX}$. The scoping follows the rules of conventional programming languages. In contrast to registers, the number of variables is limited only by the memory available.

The implementation for $\epsilon\chi\text{TEX}$ has demonstrated the extensibility and configurability of the system. It has also shown that the proposed definition of variables leads to the desired results.

References

- [1] Gerd Neugebauer. Namespaces for $\epsilon\chi\text{TEX}$. In Volker RW Schaa, editor, *Proceedings EuroTEX 2005*, pages 67–70, 2005.

MIBIB \TeX : Reporting the experience*

Jean-Michel Hufflen

LIFC (EA CNRS 4157)

University of Franche-Comté

16, route de Gray

25030 BESANÇON CEDEX

France

hufflen (at) lifc dot univ-fcomte dot fr

http://lifc.univ-fcomte.fr/~hufflen

Abstract

This article reports how the different steps of the MIBIB \TeX project were conducted until the first public release. We particularly focus on the problems raised by reimplementing a program (BIB \TeX) that came out in the 1980's. Since that time, implementation techniques have evolved and new requirements have appeared, as well as new programs within \TeX 's galaxy. Our choices are explained and discussed.

Keywords \TeX , \LaTeX , BIB \TeX , reimplementing, reverse engineering, implementation language, program update.

Streszczenie

Artykuł omawia realizację poszczególnych kroków przedsięwzięcia MIBIB \TeX , w czasie do przedstawienia pierwszej publicznej wersji. W szczególności skupiamy się na problemach powstałych przy reimplementacji programu (BIB \TeX), powstałego w latach 80 zeszłego wieku. Od tego czasu rozwinęły się techniki implementacyjne, powstały nowe wymagania oraz nowe programy w świecie \TeX -owym. Przedstawiamy i dyskutujemy dokonane wybory.

Słowa kluczowe \TeX , \LaTeX , BIB \TeX , reimplementacja, *reverse engineering*, język implementacji, aktualizacja programu.

0 Introduction

In 2003, \TeX 's 25th anniversary was celebrated at the TUG¹ conference, held in Hawaii [1]. \LaTeX [28] and BIB \TeX [35]—the bibliography processor usually associated with the \LaTeX word processor— are more recent, since they came out in the 1980's, shortly after \TeX . All are still widely used, such longevity being exceptional for software. However, these programs are aging. Of course, recent versions have incorporated many features absent from the first versions, which proves the robustness of these systems. Nevertheless, they present some limitations due to the original conception, and a major reimplementing may be needed to integrate some modern requirements. In addition, interactive word processors have made important progress and are serious rivals, even if they do not yield typesetting of such professional quality. That is why some projects

aim to provide new programs, based on \TeX & Co.'s ideas.² A first representative example is the \LaTeX 3 project [32], a second is $\mathcal{N}\mathcal{T}\mathcal{S}$ [27].

MIBIB \TeX —for 'MultiLingual BIB \TeX '— belongs to the class of such projects. Let us recall that this program aims to be a 'better BIB \TeX ', especially regarding multilingual features. For an end-user, MIBIB \TeX behaves exactly like 'classical' BIB \TeX : it searches bibliography data base (.bib) files for citation keys used in a document and then arranges the references found, writing them to a .bbl file suitable for \LaTeX , w.r.t. a bibliography style. MIBIB \TeX is written in Scheme,³ it uses XML⁴ as a

² Concerning \TeX , an additional point is that \TeX 's development has been frozen by its author, Donald E. Knuth [26]. If incorporating new ideas to a 'new \TeX ' leads to a major reimplementing, the resulting program must be named differently.

³ The version used is described in [24].

⁴ EXtensible Markup Language. Readers interested in an introductory book to this formalism can consult [37].

* Title in Polish: *MIBIB \TeX : raport z doświadczeń*.

¹ \TeX Users Group.

central format: when entries of `.bib` files are parsed, they result in an XML tree. Bibliography styles taking advantage as far as possible of MIBIB \TeX 's new features are written using `nbst`,⁵ a variant of XSLT⁶ described in [15]. The stack-based `bst` language [34] used for writing bibliography styles of BIB \TeX can be used in a compatibility mode [20].

We think that the experience we have gained in developing MIBIB \TeX may be useful for other, analogous, projects. To begin, we briefly review the chronology of this development. As will be seen, this development has not been linear, and the two following sections focus on the problems we had to face. We explain how we have determined which criteria are accurate when a programming language is to be chosen for such an application. Then we show how the compatibility with ‘old’ data and the integration of modern features should be managed.

1 MIBIB \TeX 's chronology

Oct. 2000 MIBIB \TeX 's design begins: the syntax of `.bib` files is enriched with multilingual annotations. Version 1.1's prototype is written using the C programming language and tries to reuse parts of ‘old’ BIB \TeX 's program as far as possible.

May 2001 The first article about MIBIB \TeX is [9]. Later, the experience of developing MIBIB \TeX 's Version 1.1 is described in [10].

May 2002 After discussions with participants at the EuroBach \TeX conference, we realise that the conventions for bibliography styles are too diverse, even if we consider only those of European countries. We realise that this first approach is quite unsuitable, without defining a new version of the `bst` language. So we decide to explore two directions. First, we develop a questionnaire about problems and conventions concerning bibliography styles used within European countries. Second, we begin a prototype in Scheme implementing the `bst` language [11]. Initially, this prototype is devoted to experiments about improving `bst` in a second version, 1.2.

Jan. 2003 Version 1.2 is stalled. The new version (1.3) is based on XML formats. The `nbst` language is designed and presented at [12, 13]. We explain in [14] how the results of our questionnaire have influenced this new direction.

Feb. 2004 It appears to us that MIBIB \TeX should be developed using a very high-level program-

ming language, higher than C. So we consider again the prototype in Scheme that we sketched in 2002. SXML⁷ [25] is chosen as the representation of XML texts in Scheme. Some parts of MIBIB \TeX are directly reprogrammed from C to Scheme. As for the other parts, this prototype is a good basis for much experiment [16].

Nov. 2004 The version written in C is definitely dropped, whereas the version in Scheme is modified to improve efficiency; it becomes the ‘official’ MIBIB \TeX [18].

Sep. 2005 We decided to freeze MIBIB \TeX 's design and concentrate only on finishing programming. Many Scheme functions are rewritten in conformity to SRFIs⁸ [39].

May 2006 A working version is almost finished, except for the interface with the `kpathsea` library.

May 2007 Public availability of MIBIB \TeX 's Version 1.3.

Let us also explain that MIBIB \TeX is not our only task. As an Assistant Professor in our university, we teach computer science, and participate in other projects. As a consequence, MIBIB \TeX 's development has been somewhat anarchic: we hardly worked on it for two or three months, put it aside for one or two months, and so on. Last, we have supervised some student projects regarding graphical tools around MIBIB \TeX [2, 8], programmed using Ruby [31], but concerning the development of the MIBIB \TeX program itself, we have done it alone.

2 Choice of an implementation language

There are several programming paradigms: imperative, functional, and logic programming. There are also several ways to implement a programming language: interpretation and compilation. Some paradigms are more appropriate, according to the domain of interest. Likewise, some interpreted languages are more appropriate if you want to program a prototype quickly and are just interested in performing some experiment.⁹ But compiled languages are often preferable if a program's efficiency is crucial. In addition, the level of a programming language has some influence on development: in a high-level language, low-level details of structures' implementation do not have to be made explicit, so

⁷ Scheme implementation of XML.

⁸ Scheme **R**equests for **I**mplementation, an effort to coordinate libraries and other additions to the Scheme language between implementations.

⁹ Such is the case for the two graphical tools around MIBIB \TeX programmed in Ruby by our students [2, 8].

⁵ New Bibliography **S**Tyles.

⁶ e**X**tensible Stylesheet Language Transformations, the language of transformations used for XML documents [44].

development is quicker, and the resulting programs are more concise, nearer to a mathematical model.

In addition to these general considerations, let us recall that we aim to replace an existing program by a new one. This new program is supposed to do better than the ‘old’ one. ‘To do better’ may mean ‘to have more functionalities, more expressive power’, but for sake of credibility, it is desirable for the new program to be as efficient as the ‘old’ one. Let us not forget that \TeX and $\text{BIB}\TeX$ are written using an old style of programming — more precisely, a monolithic style used in the 1970’s–1980’s — based mainly on global variables, without abstract data types. Choosing a language implemented efficiently is crucial: as a counter-example, $\mathcal{N}\mathcal{T}\mathcal{S}$, written using `Java`, has been reported 100 times slower than \TeX [42, § 5]. That is why we wrote MIBIB \TeX ’s first version using `C`, because of its efficiency. In addition, this language is portable to most operating systems. And to make our program modular, we defined precise rules for naming procedures [10, § 3]. But two problems appeared.

First, MIBIB \TeX ’s development has not been a daily task, as mentioned above. Even if we are personally able to program large applications in `C`, it is difficult to put aside a `C` program and resume it later: from this point of view, `C` is not a very high-level language. Besides, let us not forget that we are working within an open domain, as natural languages are. A change may be needed because of new features concerning languages that had not yet been integrated into MIBIB \TeX ’s framework. The higher the level, the more easily such a change can be applied.

Second, we want end-users of MIBIB \TeX to be able to influence the behaviour of this program. For example, many $\text{BIB}\TeX$ users put \LaTeX commands inside values associated with fields of `.bib` files, in order to increase their expressive power within bibliographical data. These users should be able to specify how to handle such commands when `.bib` files are converted into XML trees. In particular, this is useful if MIBIB \TeX is used to produce outputs for word processors other than \LaTeX [21]. How to do that in `C`, without defining a mini-language to express such functions? In this case, using a script language is a better choice . . . provided that this language is efficient. Another choice is a `Lisp`¹⁰ dialect, as in Emacs [40]: end-users can customise Emacs’ behaviour by writing expressions using the Emacs `Lisp` language [30]. This choice is homogeneous: the entire Emacs editor is expressed in Emacs `Lisp`, excepting for the

implementation of low-level functionalities.

Finally, our choice was Scheme, the modern dialect of `Lisp`. We confess that we are personally attracted by functional programming languages, because they can abstract procedures as well as data: in this sense, they are very high-level programming languages. Concerning Scheme, it seems to us to be undebatable that it has very good expressive power, and takes as much advantage as possible of lexical scoping. In addition, it allows some operations to be programmed ‘impurely’, by side effects, as in imperative programming, in order to increase efficiency. However, we use this feature parsimoniously, on local variables, since it breaks the principles of functional programming. We have defined precise rules for naming variables, as we did in `C` for the first version, in order to emphasise the modular decomposition of our program [19]. Last but not least, Scheme programs may be interpreted — when software is being developed — or compiled, in which case they are more efficient. As an example of a good Scheme implementation, `bigloo` [38] compiles Scheme functions by transforming them into `C` functions, then these `C` functions are compiled, in turn.

If we compare the implementations in `C` and Scheme, the latter is better, as expected from a very high-level programming language. But programming an application related to \TeX using a language other than `C` reveals a drawback: the `kpathsea` library [3] is written in `C`. Let us recall that `kpathsea` implements functions navigating through the TDS¹¹ [43]. In particular, such functions localise the files containing the specification of a class for a \LaTeX document or a bibliography style when $\text{BIB}\TeX$ runs. If there is a compatibility mode, for ‘old’ bibliography styles written in `bst`, the functions of this compatibility mode should be able to localise such files too. Likewise, ‘new’ bibliography styles written in `nbst`, should be localised by means of an analogous method. This implies that the language — or, at least, an implementation of the language — used for our software includes an interface with `C`.

Of course, what we expose above proceeds from general considerations. After all, we do not know if $\text{BIB}\TeX++$ [4] — a successor of $\text{BIB}\TeX$ based on `Java`, with bibliography styles also written in `Java` — is much less efficient than $\text{BIB}\TeX$. This may not be the case. The advantages of script languages in such development appear if we consider `Bibulus` [46], another successor of $\text{BIB}\TeX$, written using `Perl`.¹² It

¹¹ \TeX Directory Structure.

¹² Practical Extraction Report Language. A didactic introduction to this language is [45].

¹⁰ `LIS`t `P`rocessor.

has developed more quickly than `MLBIBTEX`, but is ‘less’ multilingual and uses `BIBTEX` when it runs. That is, `Bibulus` does not replace `BIBTEX` wholly, as `MLBIBTEX` attempts to do. In addition, there is an example where the need of a programming language at a higher level than C appeared: the project of moving Ω — a successor of `TEX` — into a C++ platform [36].

We personally think that an implementation of `NS` in `Common Lisp` [41] — what was planned initially — would have been preferable. As mentioned in [47], the object-oriented features of `Common Lisp` (`CLOS`¹³) have been added to the language’s basis — as C++ object-oriented functionalities are added on top of C — but the language itself is not actually object-oriented. In [47], this point is viewed as a drawback. First, we personally think that not everything is an object, from the point of view of conceiving ideas. Second, `Common Lisp`, even if it is a functional programming language, allows some operations to be performed more efficiently by means of side effects, like Scheme.¹⁴ But `Common Lisp`’s standard does not specify an interface with C, as Scheme does, although some implementations provide this service. However, we personally prefer Scheme: it is simpler and more modern.

3 Choice of strategy

3.1 Languages

`TEX` & Co. have been wonderful programs since the date they came out. Although they behave very nicely, the syntaxes are quite archaic. `TEX`’s is not homogeneous — although `LATEX 2ε` and `LATEX 3` [32] try to correct this point — for example, different delimiters are used to change size (`{\small ...}`) or face (`\textbf{...}`). `BIBTEX`’s syntax suffers from lack of expressive power: for example, the only way to put a brace within a field’s value is to give its code number by `\symbol{...}`. ‘Semantically’, `TEX`’s language provides many intelligent features, as mentioned in [6], but does not meet a modern style of programming. Likewise, `.bib` files’ syntax can express only ‘verbatim’ values, except for some ‘tricks’ like inserting ‘-’ characters for a range of page numbers. The specification of structured values like person or organisation names is easy for simple cases, but quickly becomes obscure in more complicated cases [22].

In addition, new syntactic sugar may be needed to meet some new requirements. As an example, [23]

points out that the arguments of some macros — e.g., `\catcode` — are not easily parseable. As another example, the `ConTEXt` format [7] implements a homogeneous expression of setup commands, by means of a ‘`key=value`’ syntax:

```
\setuplayout[backspace=4cm,topspace=2.5cm]
```

Nevertheless, is it reasonable to add more and more syntactic sugar to such old-fashioned syntax? Would the definition of new languages not be preferable? Of course, the present languages of `TEX` and `BIBTEX` will still remain to be used, due to the huge number of files using them and developed by end-users. But if a new language is designed, it should become the usual way to deal with the new program. Of course, end-users will have to get used to the new language. But that can be done progressively and synergy between developers and users may cause this new language to be improved if need be.

In addition, let us remark that in our case, the new language for bibliography styles (`nbst`) is close to `XSLT`, so we think that users familiar with the former can get used to the latter easily.

3.2 New services

Now it is admitted that composite tasks are not to be done by a monolithic program, but by means of a cooperation among several programs. From this point of view, the cooperation between `LATEX` and `BIBTEX` is exemplary. But `BIBTEX` is too strongly related to `LATEX`. `BIBTEX` can be used to build bibliographies for `ConTEXt` documents, but only because this word processor belongs to the `TEX` family. On the contrary, writing a converter from `BIBTEX` to `HTML`¹⁵ by means of the `bst` language is impossible without loss of quality: for example, the unbreakable space character is represented by ‘~’ — as in `TEX` — when names are formatted [22], and this convention cannot be changed.¹⁶ We see that such problems can be avoided by considering an XML-like language as a central format. In our case, generating bibliographies according to formats other than `LATEX`’s should be easy since the `LATEX` commands end users put into `.bib` files are removed when these files are parsed. This point is detailed in [17, 21].

4 Conclusion

Last but not least, we have enjoyed designing and implementing `MLBIBTEX`, even if this development backtracked several times. In addition, we think

¹⁵ **HyperText Markup Language**. Readers interested in an introduction to this language can refer to [33].

¹⁶ In fact, there are such converters, an example being `BIBTEX2HTML` [5], written using `Objective Caml` [29], a functional programming language.

¹³ **Common Lisp Object System**.

¹⁴ **Emacs Lisp**, too, and the components of the `Emacs` editor largely use this feature.

that this development shows the difficulties related to such a task. Two parts have to be managed in parallel. The first part is *reverse engineering*, that is, guessing the concept from the program. The second: enlarging what already exists. In comparison with ‘classical’ development of a new program from scratch, tests concerning the compatibility mode are easy to perform: we can simply compare what is given by the two programs, the ‘old’ one becoming an oracle. But reaching a homogeneous concept is not obvious if we want to keep backward compatibility. Nevertheless, we hope that we have done some satisfactory work.

5 Acknowledgements

Many thanks to Jerzy B. Ludwiczowski, who has written the Polish translation of the abstract, and to the proofreaders: Karl Berry, Barbara Beeton.

References

- [1] William ADAMS, ed.: *TUG 2003 Proceedings*, *TUGboat*, Vol. 24:1. July 2003.
- [2] Cédric BASSETTI and Christian BON: *Interactive Specification of bibliography styles for MIBIB \TeX* . Report of student project. University of Franche-Comté. May 2006.
- [3] Karl BERRY and Olaf WEBER: *Kpathsea library*. <http://tug.org/kpathsea/>.
- [4] Emmanuel DONIN DE ROSIÈRE: *From Stack Removing in Stack-Based Languages to BIB \TeX ++*. Master’s thesis, ENSTB, Brest. 2003.
- [5] Jean-Christophe FILLIÂTRE and Claude MARCHÉ: *The BIB \TeX 2HTML Home Page*. June 2006. <http://www.lri.fr/~filliatr/bibtex2html/>.
- [6] Jonathan FINE: “ \TeX as a Callable Function”. In: *Euro \TeX 2002*, pp. 26–30. Bachotek, Poland. April 2002.
- [7] Hans HAGEN: *Con \TeX t, the Manual*. November 2001. <http://www.pragma-ade.com/general/manuals/cont-enp.pdf>.
- [8] Stéphane HENRY and Jérôme VOINOT: *Interface for MIBIB \TeX . Getting Bibliographical Entries Interactively*. Report of student project. University of Franche-Comté. May 2005.
- [9] Jean-Michel HUFFLEN : « Vers une extension multilingue de BIB \TeX ». *Cahiers GUTenberg*, Vol. 39–40, p. 23–38. In *Actes du Congrès GUTenberg 2001*, Metz. Mai 2001.
- [10] Jean-Michel HUFFLEN: “Lessons from a Bibliography Program’s Reimplementation”. In: Mark VAN DEN BRAND and Ralf LÄMMEL, eds., *LDTA 2002*, Vol. 65.3 of *ENTCS*. Elsevier, Grenoble, France. April 2002.
- [11] Jean-Michel HUFFLEN: *Interaktive BIB \TeX -Programmierung*. DANTE, Herbsttagung 2002, Augsburg. Oktober 2002.
- [12] Jean-Michel HUFFLEN: *Die neue Sprache für MIBIB \TeX* . DANTE 2003, Bremen. April 2003.
- [13] Jean-Michel HUFFLEN: “Mixing Two Bibliography Style Languages”. In: Barrett R. BRYANT and João SARAIVA, eds., *LDTA 2003*, Vol. 82.3 of *ENTCS*. Elsevier, Warsaw, Poland. April 2003.
- [14] Jean-Michel HUFFLEN: “European Bibliography Styles and MIBIB \TeX ”. *TUGboat*, Vol. 24, no. 3, pp. 489–498. Euro \TeX 2003, Brest, France. June 2003.
- [15] Jean-Michel HUFFLEN: “MIBIB \TeX ’s Version 1.3”. *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.
- [16] Jean-Michel HUFFLEN: “A Tour around MIBIB \TeX and Its Implementation(s)”. *Biuletyn GUST*, Vol. 20, pp. 21–28. In *Bacho \TeX 2004 conference*. April 2004.
- [17] Jean-Michel HUFFLEN: “MIBIB \TeX : Beyond \LaTeX ”. In: Apostolos SYROPOULOS, Karl BERRY, Yannis HARALAMBOUS, Baden HUGHES, Steven PETER and John PLAICE, eds., *International Conference on \TeX , XML, and Digital Typography*, Vol. 3130 of *LNCS*, pp. 203–215. Springer, Xanthi, Greece. August 2004.
- [18] Jean-Michel HUFFLEN: *Beschreibung der MIBIB \TeX -Implementierung mit Scheme*. DANTE 2004, Herbsttagung, Hannover. Oktober 2004.
- [19] Jean-Michel HUFFLEN: “Implementing a Bibliography Processor in Scheme”. In: J. Michael ASHLEY and Michel SPERBER, eds., *Proc. of the 6th Workshop on Scheme and Functional Programming*, Vol. 619 of *Indiana University Computer Science Department*, pp. 77–87. Tallinn. September 2005.
- [20] Jean-Michel HUFFLEN: “BIB \TeX , MIBIB \TeX and Bibliography Styles”. *Biuletyn GUST*, Vol. 23, pp. 76–80. In *Bacho \TeX 2006 conference*. April 2006.
- [21] Jean-Michel HUFFLEN: “MIBIB \TeX Meets Con \TeX t”. *TUGboat*, Vol. 27, no. 1,

- pp. 76–82. EuroTeX 2006 proceedings, Debrecen, Hungary. July 2006.
- [22] Jean-Michel HUFFLEN: “Names in BibTeX and MIBibTeX”. *TUGboat*, Vol. 27, no. 2, pp. 243–253. TUG 2006 proceedings, Marrakesh, Morocco. November 2006.
- [23] David KASTRUP: “Designing an Implementation Language for a TeX Successor”. In: *Proc. EuroTeX 2005*, pp. 71–75. February 2005.
- [24] Richard KELSEY, William D. CLINGER, Jonathan A. REES, Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIK, 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: “Revised⁵ Report on the Algorithmic Language Scheme”. *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.
- [25] Oleg E. KISELYOV: *XML and Scheme*. September 2005. <http://okmij.org/ftp/Scheme/xml.html>.
- [26] Donald Ervin KNUTH: “The Future of TeX and METAFONT”. *TUGboat*, Vol. 11, no. 4, pp. 489. December 1990.
- [27] Joachim LAMMARSCH: “The History of $\mathcal{N}\mathcal{T}\mathcal{S}$ ”. In: *EuroTeX 1999*, pp. 228–232. Heidelberg (Germany). September 1999.
- [28] Leslie LAMPORT: *L^AT_EX: A Document Preparation System. User’s Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.
- [29] Xavier LEROY, Damien DOLIGEZ, Jacques GARRIGUE, Didier RÉMY and Jérôme VOUILLOIN: *The Objective Caml System. Release 0.9. Documentation and User’s Manual*. 2004. <http://caml.inria.fr/pub/docs/manual-ocaml/index.html>.
- [30] Bill LEWIS, Dan LALIBERTE, Richard M. STALLMAN and THE GNU MANUAL GROUP: *GNU Emacs Lisp Reference Manual*. <http://www.gnu.org/software/emacs/elisp-manual/>.
- [31] Yukihiko MATSUMOTO: *Ruby in a Nutshell*. O’Reilly. English translation by David L. Reynolds, Jr. November 2001.
- [32] Frank MITTELBACH and Rainer SCHÖPF: “Towards L^AT_EX 3.0”. *TUGboat*, Vol. 12, no. 1, pp. 74–79. March 1991.
- [33] Chuck MUSCIANO and Bill KENNEDY: *HTML & XHTML: The Definitive Guide*. 5th edition. O’Reilly & Associates, Inc. August 2002.
- [34] Oren PATASHNIK: *Designing BibTeX Styles*. February 1988. Part of the BibTeX distribution.
- [35] Oren PATASHNIK: *BibTeXing*. February 1988. Part of the BibTeX distribution.
- [36] John PLAICE and Paul SWOBODA: “Moving Omega to a C++-Based Platform”. *Biuletyn Polskiej Grupy Użytkowników Systemu TeX*, Vol. 20, pp. 3–5. In *BachTeX 2004 conference*. April 2004.
- [37] Erik T. RAY: *Learning XML*. O’Reilly & Associates, Inc. January 2001.
- [38] Manuel SERRANO: *Bigloo. A Practical Scheme Compiler. User Manual for Version 2.9a*. December 2006.
- [39] *Scheme Requests for Implementation*. February 2007. <http://srfi.schemers.org>.
- [40] Richard M. STALLMAN: *GNU Emacs Manual*. January 2007. <http://www.gnu.org/software/emacs/manual/>.
- [41] Guy Lewis STEELE, JR., Scott E. FAHLMAN, Richard P. GABRIEL, David A. MOON, Daniel L. WEINREB, Daniel Gureasko BOBROW, Linda G. DEMICHIEL, Sonya E. KEENE, Gregor KICZALES, Crispin PERDUE, Kent M. PITMAN, Richard WATERS and Jon L WHITE: *COMMON LISP. The Language. Second Edition*. Digital Press. 1990.
- [42] Philip TAYLOR, Jiří ZLATUŠKA and Karel SKOUPÝ: “The $\mathcal{N}\mathcal{T}\mathcal{S}$ Project: From Conception to Implementation”. *Cahiers GUTenberg*, Vol. 35–36, pp. 53–77. May 2000.
- [43] TUG Working Group on a TeX Directory Structure: *A Directory Structure for TeX Files*. <http://tug.org/tds>.
- [44] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [45] Larry WALL, Tom CHRISTIANSEN and Jon ORWANT: *Programming Perl*. 3rd edition. O’Reilly & Associates, Inc. July 2000.
- [46] Thomas WIDMAN: “Bibulus—a Perl XML Replacement for BibTeX”. In: *EuroTeX 2003*, pp. 137–141. ENSTB. June 2003.
- [47] Jiří ZLATUŠKA: “ $\mathcal{N}\mathcal{T}\mathcal{S}$: Programming Languages and Paradigms”. In: *EuroTeX 1999*, pp. 241–245. Heidelberg (Germany). September 1999.

Writing (L^A)T_EX documents with AUCT_EX in Emacs

David Kastrup
dak (at) gnu dot org

Abstract

At the time of the abstract deadline, several pretest versions of Emacs 22 have been made available, and the final release is even more imminent than the last few years. However, most GNU/Linux distributions already have made developer versions of Emacs available as snapshots. Users meeting their typesetting needs mostly with L^AT_EX will profit from moving to such versions from the rather ancient Emacs 21.4 because of extensive improvements of the provided desktop and editing environment.

A number of newly supported version control systems, thumbnail-supported browsing of directories with graphics files, considerably improved Unicode support for editing, desktop interaction and input, syntax highlighting activated by default, new ports for Windows, Mac OS X and GTK+ using the native toolkits for graphic support and toolbars and providing a native, well integrated look for those desktop environments, transparent access to files accessible with `su`, `sudo`, `ssh` and other shell accounts: those provide, among numerous improved details and fixes, quite a bit of progress for using Emacs as a work environment.

Focusing on the creation of L^AT_EX documents, the AUCT_EX editing package maintained by the speaker is the most extensively used editing solution for T_EX and Emacs, providing previewed material integrated into the source code window with `preview-latex`, support of source specials and the `pdfsync` package for lowering the barrier between source code and final output, and delivering a number of ways for formatting and organizing the source code. Syntax highlighting and folding of various constructs and comments render source code more manageable. A specialized mode for editing `.dtx` files considerably supports the labors of T_EX programmers. For managing cross references and bibliographical citations in L^AT_EX, the RefT_EX package provides convenient support.

1 Tutorial

It will be shown how to get to an installed version of Emacs and what to do with it.

2 Sources

The abstract explains what this is all about. Let us just mention the sources where you can get Emacs/AUCT_EX combinations. At the current point of time, the pretest release numbers have reached 22.0.97. By the time of the conference, Emacs 22.1 might well be released. In any case, here is the availability at the time of this writing:

First stop Try the download page from AUCT_EX at <http://www.gnu.org/software/auctex>.

Debian/Ubuntu The packages `auctex` as well as `emacs-snapshot-gtk` are available in the

usual repositories for those distributions. Current versions should be 22.0.95 and 11.84 or later.

Mac OS X CarbonEmacs (<http://homepage.mac.com/zenitani/emacs-e.html>) comes with AUCT_EX.

MS Windows Download a precompiled Emacs with AUCT_EX from the AUCT_EX download page.

Fedora Download the AUCT_EX RPM from the AUCT_EX download page. Get Emacs 22 from <http://people.redhat.com/coldwell/emacs/>.

SUSE Download the AUCT_EX RPM from the AUCT_EX download page. There seems to be no good source for a precompiled Emacs 22. Compile your own (downloading the source from <http://alpha.gnu.org/gnu/emacs/pretest>) or stay with 21.4 (ugh).

LyX: An editor not just for secretaries

Tomasz Łuczak

Katowice, Poland

tlu (at) technodat dot com dot pl

Abstract

The article presents a less known but nonetheless interesting editor named LyX, which can be used not only for mundane secretarial tasks but also more difficult jobs.

1 What is LyX?

One could say that an editor is an editor, but LyX is a little bit different.

The basic difference with other TeX editors lies in that LyX does not display TeX commands, it writes files in its own format, and the text in the editor window is pre-formatted (Fig. 1).

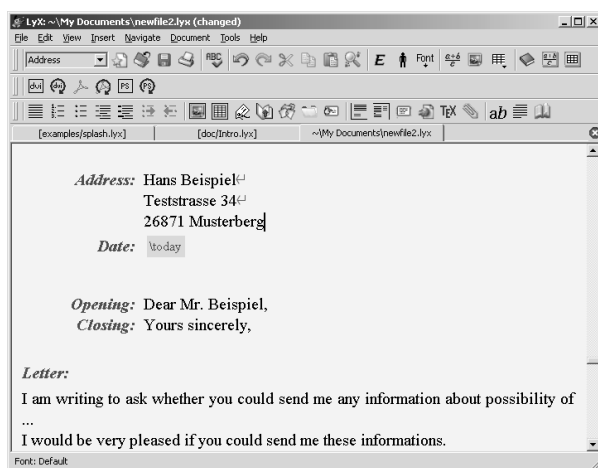


Figure 1: The LyX editor window

Text preformatting does not make LyX a WYSIWYG, i.e., *what you see is what you get* editor but a so-called WYSIWYM, i.e., *what you see is what you mean* editor. That what we see is what we want to achieve, thus in what way TeX is going to typeset it is a secondary issue — TeX surely will do its best. In the long run the WYSIWYG mode is tiresome and takes the author’s attention from the content. With LyX the text is preformatted only to mark out (sub)titles, font size changes, lists or tables.

Options are available to set the basic text font and its size as well as colors. This is to facilitate the writing process. The default light rose background does not strain the eyes.

The editor comes with extensive documentation and an excellent tutorial, so one can quickly reach

the stage of making conscious use of the program.

LyX is available for Linux, MacOSX and MS Windows. It now offers Unicode support with UTF-8 encoded input and XeTeX processing.

The LyX home page is <http://www.lyx.org>.

2 Working with LyX

A new LyX document is created in two ways: by selecting from one of a set of templates or opening an empty document. An empty document is not empty — it has a default preamble. In document settings the class is chosen from a list; one may choose additional class options, page geometry parameters and thus “click-out” the preamble.

The document settings area also offers fields for entering one’s own “magic words” to be placed in the preamble.

LyX allows for the creation of document templates: new documents can be composed from such templates. They are just ordinary LyX documents in which a class with its parameters is defined and sample text is filled in. Templates are convenient not only for more complicated documents like presentations or articles in predefined journal styles but also for letters or ordinary papers because they automate and minimize standard actions.

Titles, lists, and most other environments are selected from a list located to the left on the tool bar, just below the menu. Font or font size changes, paragraph settings are conducted through dialog windows. Many common environments and editing commands are available through keyboard shortcuts which immensely speeds up the writing.

Labels and references, index items, tables of contents, tables, images, inserts, minipages, quotations, footnotes and so on, i.e., all basic document elements, are available from the menu or tool bars with one or two mouse clicks.

LyX offers convenient tools for table editing, available from the tools menu or by right-clicking directly on the table. Changes of table layout, justi-

fication, joining cells in rows, borders and so on are easy to do with L_AT_EX.

Inserting images into the text is similarly uncomplicated. One can specify if the image should be shown in the editor window and at what size. Inserts with pictures or tables might be folded to minimize distraction.

After the text is entered, it is time to compile the document, which in L_AT_EX terminology is called “viewing”, as compilation is automatically followed by viewing of the compiled document. This is convenient and nice, because L_AT_EX compiles until all references become up-to-date. If a bibliography or indexes are present in the document, the relevant programs will also be called in the proper order.

If the document contains an error, L_AT_EX will display a window with the list of errors. This allows one to navigate through the document to the places which T_EX indicates. There shouldn’t be many errors; for example, L_AT_EX itself enters names of the environments we choose from the list.

To facilitate navigation and approximate the result, titles and lists are numbered automatically and the table of contents is created. A nice feature is navigation bookmarks, of which one can create up to five.

Change tracking, indicated by the use of color, is a tool which cannot be overestimated. Changes made in the document might be accepted or rejected. Moreover, version control, based on RCS, is built into L_AT_EX.

The comfort of editing is increased by the ability to open several buffers with one document as well as the availability of session management.

3 More advanced features

The authors of the editor did not forget about mathematical expressions, which are displayed in the editor’s window and are comfortably editable. The math symbol panel allows for easy access to needed symbols.

Math expressions entered into a L_AT_EX document can be passed to one of the supported computer algebra systems (Maxima, Octave, Mathematica and Maple). The calculated result is inserted into the document.

A serious article requires a bibliography. Here also, L_AT_EX does not fall short, allowing the use of a bibliography database. Unfortunately, the database has to be prepared outside of L_AT_EX. Some bibliography database management programs (e.g., TkBIB_TE_X and pybibliographer) communicate with L_AT_EX through pipes.

If the need arises to enter a L^AT_EX command,

this can be done by employing T_EX code inserts. We are thus allowed to enter code which will not be interpreted by L_AT_EX, but passed verbatim.

When working with large documents, it is convenient to divide them into smaller parts. Although each L_AT_EX document has its own preamble, L_AT_EX “sees” only the content when incorporating parts into the main document. This allows for separate compilation of parts and of the whole. Also, both L^AT_EX and text files can be included into the main document.

If we arrive at a stage where L_AT_EX does not suffice, we can convert the document from the L_AT_EX format to the T_EX format. The resulting files are readable enough to be of further use.

4 Peculiarities of L_AT_EX

For compilation, L_AT_EX creates a temporary directory to which all converted document files are written and to which all related files, e.g., images, are copied. One should remember that if one wants to have the resulting document in the same directory as the L_AT_EX document one should export it, otherwise the resulting document will not be available after the closing of the document or of the editor. L_AT_EX offers exports to all the usual formats: DVI, PS, PDF, TXT, and also to HTML formats. Other formats can be supported provided the appropriate converters are available.

L_AT_EX cooperates with the following spell checking programs: aspell, ispell and hspell. Unfortunately, spell checking does not function while typing occurs, it has to be activated manually. Spell checking starts from the current cursor position.

L_AT_EX uses document classes in a peculiar way. The editor allows the use of only those classes which are available with the T_EX installation and which have their own L_AT_EX *.layout configuration files. For most of the popular classes (standard classes, mwcls, memoir, koma-script, beamer and about a hundred others) such files exist. The configuration files contain information about class options and the way the environments are presented by the editor. All the environments which we want to be available on the list should be defined in the configuration file.

The editor automatically configures itself during its first run by locating all programs it needs and checking the availability of classes.

5 Final remarks

The most important merit of L_AT_EX is that it opens the T_EX world to those who have minimal, or none whatsoever, knowledge of it.

Happy L_AT_EXing!

Automated DVD menu authoring with pdfL^AT_EX

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) cs dot bme dot hu
<http://www.inf.bme.hu/~pts/>

Abstract

dvdauthor is an excellent low-level free tool for video DVD authoring on Unix systems. However, it doesn't provide a convenient way for drawing the menu background and buttons. We present dvdmenuauthor, a collection of scripts for automated DVD authoring with menus. dvdmenuauthor uses pdfL^AT_EX macros for menu composition, Xpdf for menu rendering, and dvdauthor for DVD filesystem authoring.

1 Background

DVD-Video [8] is today's most popular home entertainment video format. Video shops and video rental services used to provide films on VHS cassettes in the 1990s, but now they offer DVD discs almost exclusively. DVD not only provides cheaper reproduction costs, better video and audio quality than VHS, and multiple camera angles, audio tracks and subtitles, but it also has an advanced, programmable (but optional) navigation facility called DVD menus (or DVD extras).

The remote control of a DVD player device has several menu buttons (such as *menu*, *top menu*, *audio*, *subtitle* and *angle*), which, when pressed, suspend playback and jump to a menu. A menu is a single-page interactive part of the DVD, designed and programmed by the DVD creator. It can be animated (possibly in a loop), and it can have audio as well. A menu has several on-screen buttons, one of them being highlighted. The arrow buttons (*up*, *right*, *down* and *left*) on the remote control can be used move the highlight, or, when the DVD is played on a computer, the highlight is moved to the button under the mouse. The *enter* button can be used to execute the action associated to the highlighted on-screen button. Possible actions:

- resume or start playback at a specific location;
- jump to another menu (possibly with a specific button pre-highlighted);
- change a playback-related variable (such as audio stream, subtitle language and angle);
- change an auxiliary variable (to be used later) — integer arithmetic operators are available;
- execute a conditional block (*if-then-else*).

Popular reasons for adding menus to a DVD:

- DVD menus are a good quality add-on for Hollywood-style movies. Both the visual appearance and the sound of the menu is in theme with the movie, and the first minute spent on navigating the menu (mostly in order to select the audio stream and subtitles) is now part of the fun the spectator experiences.
- If the DVD contains a lot of material (up to 8 hours are feasible using double-layer discs and lossy compression), spectators expect an order in which they can easily find the title they are looking for. Menus with thumbnail images and title captions make navigation easier. It is also possible to have multiple menus that point to the same set of titles, but in different logical order. Usually 2×2 or 3×2 thumbnails are displayed in a single menu, and such menus are linked together using buttons. Most DVD authoring software provide an automated wizard for generating thumbnailed menus of this kind.
- DVD menus make it possible to present an interactive show to the spectators, in which they can choose among 2 or 3 endings of the movie, or they can even choose in the middle how the story should advance. Of course, movie creators must record all possible storylines, which is a lot of extra work, and the capacity of the DVD disk also limits the available choices. However, it can be feasible to give the spectator 3 choice points and thus have $2 \cdot 2 \cdot 2 = 8$ storylines altogether in a 1-hour long movie.
- It is also possible to offer a trivia game (playable by the spectator) in DVD format. For example, the famous *Who wants to be a millionaire* TV

game has a DVD version [16], in which the next question is chosen randomly from about 1000 pre-recorded questions. Even the three lifelines are present. All these are programmed as a set of DVD menus.

The process of designing and creating a DVD-Video disc is called *DVD authoring*. It consists of these steps:

1. *DVD stream authoring*: The video, audio and subtitle streams are created, imported and multiplexed together to DVD-compatible MPEG-2 program stream files. The DVD standard imposes quite a lot of restrictions on the file format, the video resolution, the frame rate, the video codec, the audio codec and the audio sample rate. However, there are tools (such as the free DeVeDe [2]) which can convert any stream to a DVD-compatible stream. Most video editing tools have DVD-compatible export filters. For simple MPEG video editing, we recommend MPEG Video Wizard [17], which is not only efficient to use, but it also runs quickly enough even in virtualised environments.
2. *DVD menu authoring*: The menu background images (or animations), buttons and captions are designed, menus and titles (i.e. streams authored in the previous steps) are combined using programmed actions. This step is the integration part of the DVD authoring process, because the way individual background images, thumbnails, captions and stream files are combined together is specified in this step. If `dvdauthor` [3] is used in the next step, the details of the integration are specified in its XML project input file.
3. *DVD filesystem creation*: The various stream files and declarations are combined to a DVD-Video filesystem (with the `VIDEO_TS` folder). This is a completely automatic process (and takes about 5 minutes on modern PCs for a single-layer full DVD). On Linux, `dvdauthor` [3] is the only well-known free tool that can do the job; other programs are usually easy-to-use frontends to `dvdauthor`.
4. *DVD image creation*: An ISO image file is created automatically from the DVD filesystem. On Linux systems, it is usually done with `mkisofs` [13], with the `-dvd-video` option.
5. *DVD disc burning*: The ISO image file is automatically burnt to a DVD disc. On Linux systems, `growisofs` [10] is a convenient command-line tool to do the job. It can also combine this step with the previous one (DVD image creation),

so a multi-gigabyte temporary ISO image file doesn't have to be created.

2 Motivation

This article focuses on DVD menu authoring, i.e. adding menus and integrating DVD-Video components. It presents a solution based on the combination of `dvdauthor` XML integration and L^AT_EX typesetting. The reasons why such a solution can be useful:

- Our solution uses only free software and runs on Unix systems. We have tried several tools [3, 14, 7, 11], but we haven't found such a tool for Unix which is user-friendly, well-integrated (doesn't need a specific version of several dozen other software packages to work), reliable (no random crashes) and ready for production use (no major bugs and annoyances, no memory leaks). We decided to develop our own software, which is practical and usable for menu-based DVD authoring.
- Most popular video editing programs provide only a wizard, which streamlines creating simple menus (such as thumbnail buttons for each title), and doesn't let the user specify the exact menu structure.
- A template-based, non-WYSIWYG solution is useful for repetitive, automated menu generation, such as generating a navigable DVD slideshow from a set of images, or generating several DVDs (with different video content) using one menu theme.
- L^AT_EX provides a separation of text and design that is versatile enough for several designs to be tried (and possibly customised) for the same menu structure. Most WYSIWYG DVD menu creation tools let the user manipulate the design of one object a time. Most of them don't support requests like "let's see the same design with 10% larger buttons", and even those that support it, won't be able to adjust the spacing properly around the resized buttons. With L^AT_EX, however, those kinds of changes can be easily done with glue nodes and a little macro programming.
- T_EX can typeset textual labels of high quality. Most DVD authoring programs have very limited typographic capabilities, for example they don't support manual line breaking, line justification, automatic line breaking, pair kerning and accented characters are not available. Using L^AT_EX we get all these features.

3 Design decisions

It was our intention to use existing software if possible, and add or change things where existing software is not powerful enough. We have found that `dvdauthor`'s XML project file provides an efficient and precise way for DVD menu authoring—except for drawing the menus (and converting them to a format that `dvdauthor` understands). Thus we decided to supplement the XML project file with drawing operations, and write some scripts that extract the drawing operations, typeset the menus, render them to images, convert the images, and run `dvdauthor` to create the DVD file system. We chose \LaTeX for the markup language of drawing operations, mostly because it has powerful typesetting capabilities, and its macro language is powerful enough to implement the necessary housekeeping (e.g. which button was emitted to which page).

We wanted to keep \LaTeX programming at a minimum, because \LaTeX is not convenient for general data processing. Thus we use \LaTeX mostly for typesetting. Perl scripts generate the document to typeset from the project XML file, and Perl scripts drive the further conversion of `pdf \LaTeX` 's PDF output to images (using the `pdftoppm` tool of `Xpdf` [19]). \LaTeX macros emit some meta-information to the `.aux` file (e.g. the correspondence of PDF page numbers and DVD menu button names), which is also read by the Perl scripts.

We wanted to reuse as many \LaTeX typesetting constructs as possible, thus the style file just sets the page size, the margins and the default font size, and lets the user draw the menu with \LaTeX . We don't enforce any specific layout, any layout can be designed using \TeX boxes, glues and macro programming. However, we don't use automatic page breaks: the user has to decide in advance how many menus to have. (Automatic page breaks wouldn't fit with `dvdauthor`'s project file easily anyway.) The style file also provides some drawing primitives useful for DVD menus: colourful frames, framed buttons, single-colour buttons (of any shape) and absolute positioning.

We designed the project file syntax so that users don't have to type the same information twice, and data relationships are often expressed by putting related pieces close to each other. For example, it is possible to specify the thumbnail image file name as an attribute to the DVD `<button>` tag. The image file name will be passed as a parameter to the appropriate macro that draws the image.

The software we have written, `dvdmenuauthor`, is free to use and is available for download from [6].

4 The manual way of authoring DVDs

This section gives an introduction to DVD-Video concepts, and it also presents the pure, manual way of DVD menu authoring using `dvdauthor`. The way presented here is similar to typesetting documents with \TeX : there are a bunch of input files, most of them being plain text files written by humans, and there are some non-WYSIWYG tools, which can be applied to the input files in the correct order to produce the desired output.

4.1 DVD without menus

DVD stream authoring is beyond the scope of this paper, so we assume that the movie is already prepared in a set of DVD-compatible MPEG-2 stream files. Unfortunately, there is no validator for this file format. If there is a problem with the file (for example, the wrong audio codec is used, or the multiplexing packet size is incorrect), `dvdauthor` will usually complain, but the error message doesn't always indicate clearly the reason for the problem. The free video conversion tools `MEncoder` [12] and `FFmpeg` [9] can generate a conforming MPEG-2 stream if called with the proper parameters. See the source code of `DeVeDe` [2] for parameters to `MEncoder`.

A DVD-Video disc consists of titles and menus. A title is a stream that contains video and audio (multiple video and audio channels possible). The playback of a title can start at the beginning or at any specific position (given by a time offset from the beginning). A chapter is a logical unit within a title. DVD players usually let the user choose a title (by its number) to start playback at (not all players expose chapter boundaries within the title to the user).

The simplest, completely automatic way to create a DVD without menus is to use `dvdauthor` [3]. For example,

```
dvdauthor -o dir -t a.mpg b.mpg c.mpg
dvdauthor -o dir -t d.mpg e.mpg
dvdauthor -o dir -T
```

creates a DVD with two titles. Title 1 contains 3 chapters (from the contents of video files `a.mpg`, `b.mpg` and `c.mpg`, respectively). Title 2 contains 2 chapters (from the contents of video files `d.mpg` and `e.mpg`, respectively). For each title, files `dir/VIDEO_TS/VTS_NN_*` are created, where `NN` is the number of the title. The last command (with the `-T`) creates the table of contents (to files `dir/VIDEO_TS/VIDEO_TS.*`). Please note that the contents of the video files are copied, so running `dvdauthor` takes time and needs free disk space (about the same size as the total size of the input files). The DVD filesystem created in `dir` can be played with most media players

on Linux (e.g. Xine, VLC, Kaffeine and MPlayer). If it looks right, it can be burnt to disc:

```
growisofs -dvd-compat -Z /dev/dvd \
  -dvd-video dir/
```

4.2 DVD with menus

If menus are involved or complex settings have to be specified, then a project file (in XML format) should be prepared for `dvdauthor`, which specifies all aspects of the DVD (how chapters, titles and menus should be formed from input files; what settings should be used; what code should be executed at events like title playback beginning, title playback end and remote control button press). Then the DVD filesystem would be created by the command

```
dvdauthor -x project.xml
```

The manual page of `dvdauthor` [4] gives an excellent and concise introduction to DVD-Video concepts. However, it doesn't contain examples for complex XML projects. To get such an example, one should try some GUI DVD authoring tools (such as [11, 14]) and see what files they generate.

A DVD menu is similar to a title, with some extra interactive features, such as buttons and actions. Buttons form an extra visual layer above the menu. Each button is a rectangular image (other shapes can be specified using transparent pixels) with only a small number of colors (≤ 16). Buttons on a menu page might not overlap. Each button has 4 neighbours (*left*, *right*, *up* and *down*). A neighbour is activated when the user presses the corresponding arrow button on the remote control. When a menu is being shown, it has a current button. Only the image of the current button is drawn over the video, none of the other buttons are displayed. When another button is activated, it becomes the current button, it gets displayed (and the previous button gets hidden). When the user presses the *enter* remote control button, the code associated with the current button is executed. The code can be specified in the project XML file inside the corresponding `<button>` tag. The syntax is similar to a very small subset of C, it is documented in the manual page of `dvdauthor`.

Menus can also contain actions. An action is like a button with code (to be executed), but without a visual representation. An action is activated either by the arrow keys on the remote control (in this case, the action must be a neighbour of the current button), or by special keys (such as *angle*) on the remote control. Actions are very briefly documented; just a little information can be found in the manual page of *spumux* (a tool which is part of the `dvdauthor` suite). Actions can also be used to

jump to a different menu without the *enter* remote control button. [15] is a detailed tutorial about this. Animated thumbnails (where the thumbnail of the current button is animated) can be also be created this way. To have animated thumbnails, a separate, animated menu has to be created for each button, each menu having only one button and neighbouring actions, which jump to another menu.

Both titles and menus support executing code before the title or menu is entered (specify such code inside the `<pre>` in the `dvdauthor` XML file) and when it is left (use the `<post>` tag). The `<pre>` tag of the main menu can contain code to skip the intro video unless the disc playback has just started. This can be implemented as a conditional jump instruction. The condition should depend on a variable, which is set just before jumping to the intro.

The similarity of titles and menus implies that menus can have audio and animation. These features for menus are provided by default, since the menu background is an MPEG-2 file itself, which can contain audio, and of course can contain animation. DVD authoring applications (including `dvdmenuauthor`) usually support only still images for menus, and they take care of converting these images to MPEG-2 videos of a few seconds in length. Making the menu video loop is straightforward: the menu's `<post>` code has to be extended with a jump command that jumps to the beginning of the menu. DVD players are usually slow when jumping (partially because a seek on the DVD disc is slow), so expect about a half second of audio lag when the menu loops. Some players also ignore remote control buttons during this lag.

There are some additional concepts which are important to understand before designing a DVD structure by hand. Such concepts are: `vmgm`, `titleset` and `cell`. These are documented in the manual page of `dvdauthor` [4].

Sometimes a code snippet in a DVD program is too long. Unfortunately, `dvdauthor` doesn't always indicate this error condition properly. The solution is to split the containing menu to two or more menus, each of them containing half of the code, and jumping to each other when necessary.

Restrictions There are some restrictions on the sources and targets of direct jumps in the DVD program code. (For example, one cannot jump from one `titleset` directly to another one. Another example: in some jumps, the target menu number cannot be specified — the so-called entry point must be given.) To overcome these restrictions, an indirect jump can be used: set a variable, jump to the main menu, whose

`<pre>` code examines the variable and jumps to the desired target title or menu. `dvdauthor` provides a scarcely documented facility (the `jumpad="1"` attribute) to do this automatically. However, this might produce extra errors if the DVD program code is long. It is safer to implement the indirect jumps by hand.

4.3 Drawing the menus

All aspects of the DVD can be specified straightforwardly in the `dvdauthor` XML project file, except for the menu background image, the menu button images and the button neighbour relationships. Our aim with `dvdmenuauthor` was to automate this process as far as possible, while still leaving full control to the user. But first let's see a manual method. The visual part of a DVD menu consists of:

- *background stream*: an MPEG-2 stream with video and audio. For simple menus, the audio is silent, and the video contains a single still image repeated for a couple of seconds. (`dvdmenuauthor` supports only still images without audio.)
- *button highlight layer*: a single image with a few colours and transparency. This layer consists the union of the button images. When the DVD player displays a menu, it draws the image of the current button (taken from the button highlight layer) over the background stream. A simple button highlight layer contains only a single colour besides the transparent pixels. (`dvdmenuauthor` supports only a single colour.)
- *button select layer*: similar to the button highlight layer, but a button is drawn from here when it is activated (with *enter*). The duration that the image is displayed is just a few hundred milliseconds: it lasts until the DVD player loads the next title or menu. Usually the button highlight layer is the same as the button select layer, but with a different color.
- *button bounding boxes*: the rectangular bounding box of each button on the menu. These boxes must not overlap.
- *button neighbours*: the name of the left, right, up and down neighbour for each button. `dvdauthor` is able to infer neighbourhood relationships from bounding box coordinates.
- *button and action names*: these are used by `dvdauthor` to identify the button (or action) within the menu, in order to be able to add code to be executed when the button is selected.

It is quite cumbersome to keep all these visual elements in sync by hand when drawing the menus.

For example, if we move or resize a button, then the background stream, the button layers and the button bounding boxes have to be properly modified. `dvdmenuauthor` does all these automatically.

To find out how to assemble the visual elements to an MPEG-2 stream, the easiest way is to examine the auxiliary files generated by GUI frontends to `dvdauthor` [11, 14]. The XML syntax is explained in the manual page of `dvdauthor` and its *spumux* tool, and also in [5]. `dvdmenuauthor` contains a Perl script called *genmpeg.pl*, which can generate a DVD-compatible MPEG-2 stream from a series of still images.

5 DVD menus with `dvdmenuauthor`

5.1 A menu with thumbnails

Figure 1 shows a typical thumbnail menu in a 3×2 layout. The menu has a background, a title caption, up to 6 thumbnail buttons (now actually 5), a caption for each thumbnail, and three navigation buttons to reach other menus. For simplicity, big numbers are displayed instead of real thumbnails from the video. In the figure, thumbnail button number 1 is highlighted with an ochre rectangular frame.

Figure 2 shows how to define such a menu in `dvdauthor`'s XML project file. All the visual elements, including the background, the button layers and the button bounding boxes are encoded in the file *menu42.mpg*. Probably *spumux* was used to multiplex these visual elements to the file. The figure shows that each button has a name and a corresponding program code to execute when the button is activated. If there are any actions in the menu, they also appear as `<button>` tags here.

Figure 3 shows how to draw the same menu using `dvdmenuauthor`. It also illustrates the following features of the software:

- \LaTeX markup can be used to typeset the captions (see `\emph`).
- \TeX 's line breaking algorithm can be used (see caption of button 1).
- The visual design is separated from the menu-specific data (actual captions and thumbnail images) using templates. Only the menu-specific data is shown on the figure.
- All information needed to render a button are packed together to the `<button>` XML tag. The attributes with the `tex:` namespace are passed to \LaTeX .
- There is no need to specify button bounding boxes.

To further illustrate the magic happening, Figure 4 shows the \LaTeX code snippet generated from



Figure 1: A menu created by dvdmenuauthor

```
<pgc>
  <vob file="menu42.mpg" pause="inf" />
  <button name="e1"> g5=6; </button>
  <button name="e2"> jump menu 2; </button>
  <button name="e3"> jump menu 3; </button>
  <button name="e4"> jump menu 4; </button>
  <button name="e5"> jump menu 5; </button>
  <button name="prev"> jump vmgm menu 1; </button>
  <button name="back"> jump vmgm menu 1; </button>
  <button name="next"> jump vmgm menu 1; </button>
</pgc>
```

Figure 2: A menu definition in the dvdauthor project file

```
<pgc>
  <tex:prepage>
    \thispagebgimage{}{pal_bg_light}
    \thispagetemplate{palthumbsix}
    \menucaption{Fazekas szalagavató 1998.\ december}
  </tex:prepage>
  <vob tex:file="" pause="inf" />
  <button name="e1" tex:image="1.png" tex:caption="a szalagtűzés \emph{előtt}"> g5=6; </button>
  <button name="e2" tex:image="2.png" tex:caption="szalagtűzés"> jump menu 2; </button>
  <button name="e3" tex:image="3.png" tex:caption="osztálytáncok"> jump menu 3; </button>
  <button name="e4" tex:image="4.png" tex:caption="egyéb táncok"> jump menu 4; </button>
  <button name="e5" tex:image="5.png" tex:caption="videófelvételek"> jump menu 5; </button>
  <button name="prev" tex:dummy=""> jump vmgm menu 1; </button>
  <button name="back" tex:dummy=""> jump vmgm menu 1; </button>
  <button name="next" tex:dummy=""> jump vmgm menu 1; </button>
  <post> jump vmgm menu 1; </post>
</pgc>
```

Figure 3: A menu definition in the dvdmenuauthor project file

```

\begin{dvdmenupage}{e1,e2,e3,e4,e5,prev,back,next}
\thispagebgimage{}{pal_bg_light}
\thispagetemplate{palthumbsix}
\menucaption{Fazekas szalagavató 1998.\ december}
\begingroup\def\dvdbuttonattrXname{e1}
  \def\dvdbuttonattrXcaption{a szalagtűzés \emph{előtt}}
  \def\dvdbuttonattrXimage{1.png}
\dvdprocessbutton\endgroup
\begingroup\def\dvdbuttonattrXname{e2}
  \def\dvdbuttonattrXcaption{szalagtűzés}
  \def\dvdbuttonattrXimage{2.png}
\dvdprocessbutton\endgroup
...
\begingroup\def\dvdbuttonattrXname{next}
  \def\dvdbuttonattrXdummy{}
\dvdprocessbutton\endgroup
\end{dvdmenupage}

```

Figure 4: The L^AT_EX code snippet generated from the `dvdmenuauthor` definition

the `dvdauthor` definition on Figure 3. We can see that for each attribute with a `tex:` prefix, a macro `\dvdbuttonattrX...` is defined, and the command `\dvdprocessbutton` is called after the macro definitions for each button. The `tex:dummy` attribute is just an indicator that the button must be processed by L^AT_EX.

The macros `\menucaption` and `\dvdprocessbutton` are defined in template `palthumbsix`. They take care of the visual formatting of the menu data. `\dvdprocessbutton` has the available button names hardwired, and it formats and positions a button based on its name.

5.2 New L^AT_EX commands provided

Although `dvdmenuauthor` encourages the use of existing L^AT_EX commands, it also defines some new commands, most of them related to positioning and button typesetting.

`\thispagecolor{colorname}`

Changes the background of the current page to the specified color. Like most graphics operations in `dvdmenuauthor`, it works only with pdfL^AT_EX.

`\thispagebgimage{optionlist}{filename}`

Sets the background image for the current page. The image will be loaded by the `\includegraphics` command using the `pdftex` driver.

`\thispagetemplate{templatename}`

Selects the specified template for the current page. This means defining some macros, for example the sample template `palthumbsix` defines `\menucaption` and `\dvdprocessbutton`.

`\putat{x}{y}boxspec`

Typesets the specified material with its reference point at (x, y) . Creates a properly shifted box of size zero. Can be used for absolute positioning if called at the top of the page. Can be used with `\vbox` for last line alignment or `\vtop` for first line alignment. This command can be used in templates.

`\begin{dvdmenupage}{buttonlist}`

Renders a DVD menu page with the specified buttons (in the specified order). For technical reasons (see Subsection 6.2) each button is rendered on its own onto a separate PDF page.

`\framebox{sep-dimen}{hbox-contents}`

Similar to the built-in command `\framebox`, but allows catcode changes and verbatims in the box. Unfortunately, catcode changes don't work in general in `dvdmenuauthor`, because the `dvdmenupage` environment reads its contents to a macro for multiple rendering.

`\aliascolor{oldname}{newname}`

Copies a colour definition to be usable as a different name.

`\dvdtextbutton{name}{text}`

Typesets a button. Doesn't wrap its contents in a box, so a button can be in a middle of a paragraph, even line breaks are allowed. Each button has three forms: background, highlighted and selected. In background mode, `text` is typeset normally, in highlighted mode, it is typeset in `dvdhighlightedcolor` (without color changes and images), and in selected mode it typesets in `dvdselectedcolor`.


```
\dvdframebutton{name}{text}
```

Typesets a button inside a `\hbox`. In background mode, emits the `\hbox`. In highlighted mode, it emits an empty box surrounded by a frame (using the parameters `\dvdbuttonframesep`, `\dvdbuttonframewidth` and `dvdhighlightedcolor`). This command can be used to typeset thumbnail buttons.

```
\begin{narrowcentering}
```

Like `\begin{centering}`, but doesn't reset the natural width of `\leftskip` and `\rightskip` to zero.

5.3 Working with `dvdmenuauthor`

T_EX users are familiar with the edit–compile–preview cycle of T_EX document preparation, possibly extended with a final conversion and printing or publishing. The same cycle exists with `dvdmenuauthor`. In the edit cycle, the user edits an XML project file with L^AT_EX markup for the menus, in the compile cycle the user compiles the project files (and the media files it refers to) to a DVD image, and finally the user previews the DVD on screen, including the menus and titles. The publishing step is burning the DVD filesystem to disc.

In the edit step any text editor can be used, preferably one with XML syntax highlighting facilities. Unfortunately, L^AT_EX snippets won't be highlighted as L^AT_EX markup. The `dvdmenuauthor` contains an example project file `ex.dmp.xml`.

The compile step consists of feeding the project file to a few scripts. The *Makefile* in the `dvdmenuauthor` distribution automates this. (There is no incremental compilation support yet: the whole DVD filesystem is regenerated from scratch in each `make` run.) The following steps constitute compilation:

1. *Generating the menu L^AT_EX source file and the `dvdauthor` XML project file.* This is done by the `gendap.pl` Perl script.
2. *Compiling the L^AT_EX source file to PDF.* This is just a regular pdfL^AT_EX run. The *Makefile* runs pdfL^AT_EX twice, in case there are `\refs`.
3. *Rendering the PDF to PPM raster image files.* This is done by the excellent `pdftoppm` utility from Xpdf.
4. *Combining the PPM images to short MPEG-2 streams for the menus.* This is done by the `genspuxml.pl` Perl script, which calls another Perl script `genmpeg.pl` (to generate an MPEG-2 stream from still images), and the `spumux` tool from `dvdauthor` (to multiplex the button layers to the background). Since some image processing is done in Perl, this step can take about three seconds for each menu.

5. *Authoring the DVD filesystem.* This is just a simple `dvdauthor` run (with the `-x` option using the XML project file generated in the first step). This step might take quite a lot of time (up to 5 minutes on modern systems for a full, 4.7 GB DVD), and it needs free disk space about the same size as the sum of the input video sizes.

The recommended DVD preview application is Xine [18]. It can be installed from source in any modern Linux distribution. Although the user interface (toolbars and menus) of Xine is quite ugly, and not convenient to operate, Xine has very good keybindings, especially suitable for DVD menu navigation (a remote control panel is also available — press Alt-⟨E⟩ to make it visible). See the manual page of `xine` for the list of available keys. MPlayer is not recommended for DVD menu testing, because MPlayer doesn't support DVD menus yet. Although VLC has DVD menu support, sometimes it crashes or behaves unexpectedly. Kaffeine and Totem should also be given a try.

Since `dvdauthor` runs slowly when the videos to be put on the DVD are large, the compile step might be too slow for the user. To solve this, `dvdmenuauthor` has the `dvdmenutest.sh` shell script, which warps a `dvdauthor` project XML files so that all titles (but not menus!) are replaced with dummy short videos, so that the total video size would be small, and thus `dvdauthor` runs quickly.

The publishing step means creating an ISO image (with `mkisofs`) and/or burning it to disc (with `growisofs`). The `-dvd-video` flag of `mkisofs` must be specified in both command lines.

6 Implementation tricks

Some details of the implementation are worth mentioning, because the tricks employed can be useful in other T_EX or DVD-related projects.

6.1 Bounding box calculation

`dvdmenuauthor` calculates button bounding boxes automatically. This operation is impossible within T_EX, because there is no way to inquire about the absolute (x, y) coordinates of an item within a box.

Solution: `dvdmenuauthor` uses a bluescreen technique [1]. The L^AT_EX component emits each button on its own to a PDF page with a blue background, and a Perl script analyses the rendered page. The bounding box is the smallest rectangle on the page whose complement contains only blue pixels. A little extra housekeeping is done for identifying the button pages belonging to the same menu, because later they have to be merged again to a button highlight layer.

Limitations: It is not possible to force a bounding box larger than it appears. It is not possible to use that specific shade of blue in buttons. Fortunately, the colour is configurable.

6.2 Rendering only parts of the page

Each page has to be rendered many times: once for the background, once for each button highlight image and once for each button select image. While rendering the button highlights and selects, nothing except for the current button must be drawn. While rendering the background, the buttons must not be drawn. These requirements call for a facility which disables rendering parts of a page, but adjusting the spacing just as if they were there.

A simple solution would be to ask the user to put all visible material inside `\maybe{...}`, e.g.

```
\maybe{This} \maybe{is} \maybe{surely}
\maybe{an} \maybe{unclear} \maybe{caption}
```

But users may be annoyed by the tons of `\maybes` needed to form a paragraph with line breaks.

Solution: `dvdmenuauthor` uses PDF coordinate system transformations to move away material that is to be skipped. A coordinate translation of (10000, 10000) is used at the beginning of the page, so all material is rendered outside the visible region by default. When one needs a specific part of the page actually rendered, a translation of (-10000, -10000) is used. Since \TeX 's line breaking algorithm doesn't know about the translations (it treats them as `\specials`), this nicely works together with \TeX glues and line breaking.

Limitations: Problems might occur when material is inside a `\rotatebox` or a `\scalebox`. That is, buttons should not be rotated. It would be possible to overcome this limitation by making `\rotatebox` and friends aware of the translation.

6.3 Single-colour rendering

`\dvdtextbutton` renders the highlighted version of the button in a different colour. Colour changes are disabled within that rendering.

Solution: All colour changes can be disabled by clearing the body of the macros `\set@color` and `\reset@color`. Image inclusions have to be disabled too, because there is no way to modify the colour of raster images included in \pdfLaTeX .

6.4 International character support

Accented and other international characters must survive the XML to \LaTeX conversions, and they should be typeset by \LaTeX properly.

Solution: The character set of the XML document must be specified in the `encoding=` attribute of the `<?xml` processing instruction. This declaration is read by `dvdmenuauthor`, and the proper `\usepackage{...}{inputenc}` is emitted automatically. For clarity, `dvdmenuauthor` doesn't try to interpret the XML document as characters — its built-in XML parser just treats the project file as a sequence of ASCII-based 8-bit bytes. This ensures that accented characters don't get mangled during copying from one file to another.

7 Limitations

Sometimes the user wants pixel-accurate rendering. For example, unscaled raster images should be rendered sharply on pixel boundaries, not interpolated between half-pixels. In \TeX , the bp (PostScript big point) is used for the basis of dimension calculations, to avoid rounding errors when emitting the PDF (because PDF expresses all dimensions in bp). The `pdftoppm` tool almost always renders rules and images exactly and sharply, but sometimes it renders a PDF rule of width 4bp badly: the rule becomes 5 pixels wide in the PPM image. This issue should be investigated further, possibly modifying `Xpdf` or trying another rendering engine.

Since the button highlight layer may use only a few colours (≤ 16), it is not feasible to use antialiased text in this layer. However, if the background text is antialiased, but the highlight is not, then the highlight doesn't cover exactly the background, which looks ugly. If antialiasing is turned off entirely (as a command line option to `pdftoppm`), text will look noticeably uglier (depends on the font used). It is not possible to turn off antialiasing selectively.

`dvdmenuauthor` has been tested on full PAL (720 × 576) videos. The DVD standard allows several image sizes for both PAL and NTSC. Support for all possible sizes should be added to each tool in `dvdmenuauthor`. Other parameters (such as button neighbourhood relations) should also be made configurable.

Videos are usually rendered in a different aspect ratio than their image size. The most common aspect ratios are letterbox (4:3) and widescreen (16:9). In a full PAL DVD, letterbox implies scaling from 720 × 576 to 768 × 576, and widescreen implies scaling from 720 × 576 to 1024 × 576. Thus there is a small distortion in the letterbox case and a big distortion in the widescreen case. For practical rendering reasons, `dvdauthor` uses the image size (720 × 576) and doesn't care about the distortion made by the aspect ratio correction. Thus, when authoring widescreen DVDs, menu captions would become wider than expected.

To solve this, one could render at viewing size (thus degrading rendering quality a little) or one could condense the fonts horizontally (not easy to do in T_EX, but the pdfT_EX font expansion feature might be useful).

8 Conclusion

T_EX can be used productively for typesetting many different kinds of work: high quality (possibly international) documents, maths, presentation slides and music, to name a few. Other uses of T_EX include developing software and its documentation together, preparing web-ready books, and rearranging PDF pages. DVD menu authoring is also a new, non-standard use.

We have designed `dvdmenuauthor`, a program for project-file-driven (non-WYSIWYG) automated DVD menu authoring with powerful menu drawing facilities. It is now a free proof-of-concept implementation. DVD menus can be drawn using L^AT_EX markup to have high typographic quality output. Layout and menu data can be separated using templates. Since `dvdauthor` is used for integration, users have full freedom to create complex and/or smart menus they want. The system design (compilation and templates) makes it possible to experiment with layout changes any time in the authoring process.

`dvdmenuauthor` is not easy to start using for an average user who expects WYSIWYG and wizards. However, we hope that L^AT_EX users would find it convenient, and the software can evolve from its present proof-of-concept version to a stable, general and powerful tool like the mainstream T_EX-based programs.

References

- [1] Wikipedia article on the Bluescreen technique. <http://en.wikipedia.org/wiki/Bluescreen>
- [2] DeVeDe, a GUI program to create video DVDs suitable for home players, from any number of video formats supported by MPlayer. <http://www.rastersoft.com/programas/devede.html>
- [3] `dvdauthor`, a tool that assembles multiple mpeg program streams into a suitable DVD filesystem. <http://dvdauthor.sourceforge.net/>
- [4] The manual page of `dvdauthor`. <http://dvdauthor.sourceforge.net/doc/>
- [5] Anders Dahnielson. DVD Author Primer. December 31, 2003. <http://en.dahnielson.com/2003/12/dvd-author-primer.html>
- [6] `dvdmenuauthor`, automated DVD menu authoring with pdfL^AT_EX. <http://freshmeat.net/projects/dvdmenuauthor>
- [7] DVDStyler, a cross-platform GUI DVD authoring system. <http://www.dvdstyler.de/>
- [8] Wikipedia article on DVD-Video. <http://en.wikipedia.org/wiki/DVD-Video>
- [9] FFmpeg, a very fast command-line video and audio converter. <http://ffmpeg.mplayerhq.hu/>
- [10] DVD+RW-tools, a set of tools for burning and examining DVD discs. <http://fy.chalmers.se/~appro/linux/DVD+RW/tools/>
- [11] KMediaFactory, template-based GUI DVD authoring software. <http://freshmeat.net/projects/kmediafactory/>
- [12] MEncoder, the command-line movie encoder of the MPlayer suite. <http://www.mplayerhq.hu/>
- [13] `mkisofs`, a tool for creating CD and DVD filesystems. <http://cdrecord.berlios.de/>
- [14] 'Q' DVD-Author, a GUI frontend for `dvdauthor` and other related tools. <http://qdvdauthor.sourceforge.net/>
- [15] Samantha Lane. Switched menus with `dvdauthor`. <http://www.geocities.com/samanthalane/dvd/index.html>
- [16] Who Wants to Be a Millionaire — DVD game review. http://www.ciao.co.uk/Who_Wants_To_Be_A_Millionaire_DVD_Game__Review_5480599
- [17] Womble MPEG Video Wizard, a commercial nonlinear MPEG video editor. <http://www.womble.com/products/>
- [18] Xine, a free multimedia player. <http://xinehq.de/index.php/home>
- [19] Xpdf, an open source viewer for PDF files. <http://www.foolabs.com/xpdf/>

Graphics in L^AT_EX using TikZ

Zofia Walczak

Faculty of Mathematics and Computer Science, University of Lodz
zofiawal (at) math dot uni dot lodz dot pl

Abstract

In this paper we explain some of the basic and also more advanced features of the PGF system, just enough for beginners. To make our drawing easier, we use TikZ, which is a frontend layer for PGF.

1 Introduction

In this paper we explain some of the basic and also more advanced features of the PGF system, just enough for beginners. To make our drawing easier, we use TikZ, which is a frontend layer for PGF. The commands and syntax of TikZ were influenced by such sources as METAFONT, PSTricks, and others.

For specifying points and coordinates TikZ provides a special syntax. The simplest way is to use two T_EX dimensions separated by commas in round brackets, for example (3pt,10pt). If the unit is not specified, the default values of PGF's *xy*-coordinate system are used. This means that the unit *x*-vector goes 1 cm to the right and the unit *y*-vector goes 1 cm upward. We can also specify a point in the polar coordinate system like this: (30:1cm); this means "go 1 cm in the direction of 30 degrees".

To create a picture means to draw a series of straight or curved lines. Using TikZ we can specify paths with syntax taken from MetaPost.

2 Getting started

First we have to set up our environment. To begin with, we set up our file as follows:

```
\documentclass{article}
\usepackage{tikz}
\begin{document}
Document itself
\end{document}
```

Then we start to create pictures. The basic building block of all the pictures in TikZ is the path. You start a path by specifying the coordinates of the start point, as in (0,0), and then add a "path extension operation". The simplest one is just --. The operation is then followed by the next coordinate. Every path must end with a semicolon. For drawing the path, we use `\draw` command which is an abbreviation for `\path[draw]`. The `\filldraw` command is an abbreviation for `\path[fill,draw]`.

The rule is that all TikZ graphic drawing commands must occur as an argument of the `\tikz` command or inside a `{tikzpicture}` environment. The L^AT_EX version of the `{tikzpicture}` environment is:

```
\begin{tikzpicture}[<options>]
<environment contents>
\end{tikzpicture}
```

All options given inside the environment will apply to the whole picture.

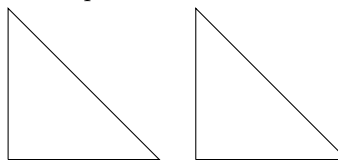
For example, to draw the triangle between the points (0,0), (0,2), (2,0) we can write:

```
\tikz\draw (0,0)--(0,2) -- (2,0)-- (0,0);
```

or

```
\begin{tikzpicture}
\draw (0,0) -- (0,2) -- (2,0)-- (0,0);
\end{tikzpicture}
```

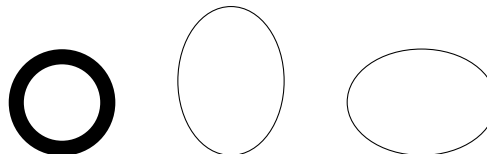
which produce:



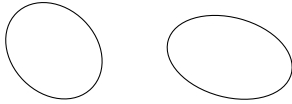
We can change the thickness of the line with the option `line width=<width>`, as in:

```
\tikz\draw[line width=2mm] (0,0) -- (0,4);
```

For drawing circles and ellipses we can use the `circle` and `ellipse` path construction operations. The `circle` operation is followed by a radius in round brackets while the `ellipse` operation is followed by two, one for the *x*-direction and one for the *y*-direction, separated by `and` and placed in round brackets. We can also add an option `rotate` or `scale` for rotating or scaling ellipse. Some examples followed by the corresponding code:

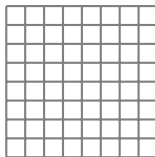


```
\tikz\draw[line width=2mm] (0,0)
circle (4ex);
\tikz\draw (0,0) ellipse (20pt and 28pt);
\tikz\draw (0,0) ellipse (28pt and 20pt);
```

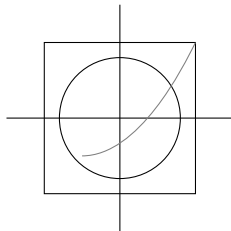


```
\tikz\draw[rotate=45] (0,0)
ellipse (16pt and 20pt);
\tikz\draw[scale=1.5,rotate=75] (0,0)
ellipse (10pt and 16pt);
```

We also have the `rectangle` path construction operation for drawing rectangles and `grid`, `parabola`, `sin`, `cos` and `arc` as well. Below are examples of using these constructions.

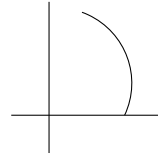


```
\begin{tikzpicture}
\draw[step=.25cm,gray,thick]
(-1,-1) grid (1,1);
\end{tikzpicture}
```



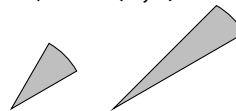
```
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle (.8cm);
\draw (-1,-1) rectangle (1,1);
\draw[gray] (-.5,-.5) parabola (1,1);
\end{tikzpicture}
```

The `arc` path construction operation is useful for drawing the arc for an angle. It draws the part of a circle of the given radius between the given angles. This operation must be followed by a triple in round brackets. The components are separated by colons. The first and second are degrees on the circle and the third is its radius. For example, `(20 : 45 : 2cm)` means that it will be an arc from 20 to 45 degrees on a circle of radius 2 cm.



```
\begin{tikzpicture}
\draw (-.5,0)--(1.5,0);
\draw (0,-.5)--(0,1.5);
\draw (1,0) arc (-25:70:1cm);
\end{tikzpicture}
```

```
\tikz\draw (0,0) arc (0:180:1cm);
```

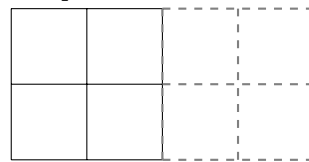


```
\tikz \draw[fill=gray!50] (4,0)-- +(30:1cm)
arc (30:60:1cm) -- cycle;
\tikz \draw[fill=gray!50] (4,0)-- +(30:2cm)
arc (30:60:1cm) -- cycle;
```

There is a very useful command `\tikzstyle` which can be used inside or outside the picture environment. With it we can set up options, which will be helpful in drawing pictures. The syntax of this command is

```
\tikzstyle<style name>+=[<options>]
```

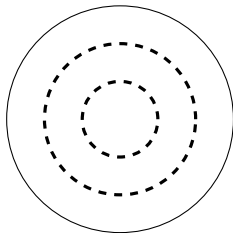
We can use it as many times as we need. It is possible to build hierarchies of styles, but you should not create cyclic dependencies. We can also redefine existing styles, as is shown below for the predefined style `help lines`:



```
\tikzstyle{my help lines}=[gray,
thick,dashed]
\begin{tikzpicture}
\draw (0,0) grid (2,2);
\draw[style=my help lines] (2,0)
grid +(2,2);
\end{tikzpicture}
```

If the optional `+` is given, it means that the new options are added to the existing definition. It is also possible to set a style using an option `<set style>` just after opening the `tikzpicture` environment.

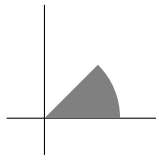
When we want to apply graphic parameters to only some path drawing or filling commands we can use the `scope` environment.



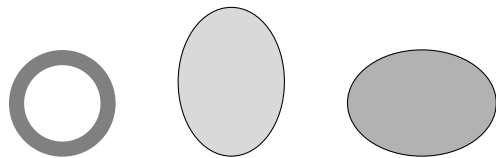
```
\begin{tikzpicture}
\begin{scope}[very thick,dashed]
\draw (0,0) circle (.5cm);
\draw (0,0) circle (1cm);
\end{scope}
\draw[thin] (0,0) circle (1.5cm);
\end{tikzpicture}
```

3 Filling with color

Using command `\fill[color]` we can fill with the given color a domain bounded by any closed curve. For closing the current path we can use `-- cycle`. For the `color` argument, we can use either name of color, for example `green`, `white`, `red`, or we can mix colors together as in `green!20!white`, meaning that we will have 20% of green and 80% of white mixed.



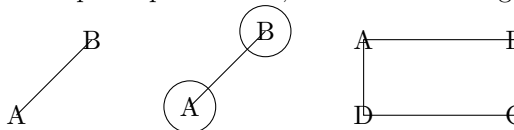
```
\begin{tikzpicture}
\draw (-.5,0)--(1.5,0);
\draw (0,-.5)--(0,1.5);
\fill[gray] (0,0) -- (1,0) arc (0:45:1cm)
-- cycle;
\end{tikzpicture}
```



```
\tikz\draw[line width=2mm,color=gray]
(0,0) circle (4ex);
\quad
\tikz\draw[fill=gray!30!white] (0,0)
ellipse (20pt and 28pt);
\quad
\tikz\draw[fill=gray!60!white] (0,0)
ellipse (28pt and 20pt);
```

4 Adding text to the picture

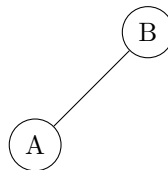
For adding text to the picture we have to add `node` to the path specification, as in the following:



```
\tikz\draw (1,1) node{A} -- (2,2) node{B};
\tikz\draw (1,1) node[circle,draw]{A} --
(2,2) node[circle,draw]{B};
\tikz\draw (0,0) node{D} -- (2,0) node{C}
-- (2,1) node{B} -- (0,1) node{A} --cycle;
```

Nodes are inserted at the current position of the path (points A and B in the first example); the option `[circle,draw]` surrounds the text by a circle, drawn at the current position (second example).

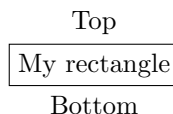
Sometimes we would like to have the node to the right or above the actual coordinate. This can be done with PGF's so-called anchoring mechanism. Here's an example:



```
\begin{tikzpicture}
\draw (1,1) node[anchor=north east,circle,
draw]{A} -- (2,2) node[anchor=south west,
circle,draw]{B};
\end{tikzpicture}
```

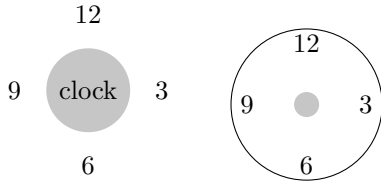
This mechanism gives us very fine control over the node placement.

For placing simple nodes we can use the `label` and the `pin` option. The `label` option syntax is: `label=[<options>]<angle>:<text>`



```
\tikz \node[rectangle,draw,
label=above:Top,label=below:
Bottom]{my rectangle};
```

When the option `label` is added to a node operation, an extra node will be added to a path containing `<text>`. It is also possible to specify the `label distance` parameter, which is the distance additionally inserted between the main node and the label node. The default is 0 pt.

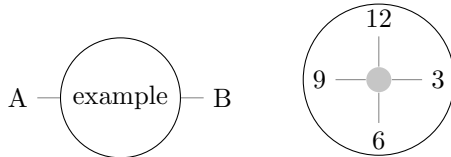


```
\tikz[label distance=2mm]
\node[circle,fill=gray!45,
label=above:12,label=right:3,
label=below:6,label=left:9]{clock};
```

The `pin` option is similar to the `label` option but it also adds an edge from this extra node to the main node. The syntax is as follows:

```
pin=[<options>]<angle>:{text}.
```

`pin distance` is an option which must be given as part of the `\tikz` command. The default is 3ex.

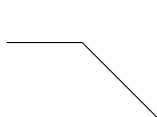


```
\tikz[pin distance=4mm]
\draw (1,1) node[circle,fill=gray!45,
pin=above:12,pin=right:3,pin=below:6,
pin=left:9]{} circle (1cm);
```

5 The plot operation

If we have to append a line or curve to a path that goes through the large number of coordinates, we can use the `plot` operation. There are two versions of `plot` syntax: `--plot <further arguments>` and `plot <further arguments>`.

The first plots the curve through the coordinates specified in `<further arguments>`; the second plots the curve by first "moving" to the first coordinate of the curve. The following example shows the difference between `--plot` and `plot`.



```
\tikz\draw (0,1) -- (1,1) --plot
coordinates {(2,0) (2,1.5)};
```



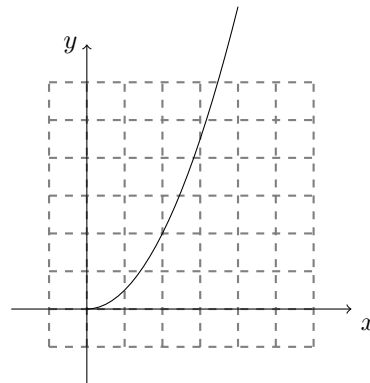
```
\tikz\draw[color=gray] (0,1) -- (1,1)
plot coordinates {(2,0) (2,1.5)};
```

6 Plotting functions

For plotting functions we have to generate many points and for that T_EX has not enough computational power, but it can call external programs that can easily produce the necessary points. TikZ knows how to call Gnuplot. In this case, the `plot` operation has the following syntax:

```
plot[id=<id>] function{formula}.
```

When TikZ encounters this operation, it will create a file called `<prefix><id>.gnuplot`, where `<prefix>` by default is the name of the `.tex` file. It is not strictly necessary to specify an `<id>`, but it is better when each plot has its own unique `<id>`. Next TikZ writes some initialization code into this file. This code sets up things such as the `plot` operation writing the coordinates into another file, named `<prefix><id>.table`.



```
\begin{tikzpicture}[domain=0:2]
\draw[thick,color=gray,step=.5cm,
dashed] (-0.5,-.5) grid (3,3);
\draw[->] (-1,0) -- (3.5,0)
node[below right] {$x$};
\draw[->] (0,-1) -- (0,3.5)
node[left] {$y$};
\draw plot[id=x] function{x*x};
\end{tikzpicture}
```

The option `samples=<number>` sets the number of samples used in the plot (default is 25) and the option `domain=<start>:<end>` sets the domain between which the samples are taken.

If you want to use the plotting mechanism you have to be sure that the `gnuplot` program is installed on your computer, and T_EX is allowed to call external programs.

References

- [1] Till Tantau, The TikZ and PGF Packages, Manual for ver. 1.09, <http://sourceforge.net/projects/pgf>

L^AT_EX vs. L^AT_EX — a modification of the logo

Grzegorz Murzynowski

Sulejówek

Poland

natror at o2 dot pl

There are at least two approaches to the T_EX, L^AT_EX, etc. logos. First, that the font used in them is a part of a logo and should not be changed, and the other, that a logo should be typeset in the same font as its context. If you choose the first approach, this article is irrelevant. In this article I suggest a slight modification of the L^AT_EX logo to make it work better with various fonts, which is relevant if we choose the second approach.

The first change is the offset of the letter *A*: in the original L^AT_EX definition it's -0.36em so it does not depend on the width of the *L* or the *A*, which differ in different fonts. In mine it's -0.57 (*width of A*) which makes it come out noticeably better.

Another change is adding one more kern between *A* and *T* if the font is slanted, that is, if its `\fontdimen1` is nonzero. The kern is $0.5\text{ex} \times$ (*slant in %*).

The original L^AT_EX definition of the logo is

```
\DeclareRobustCommand{\LaTeX}{%
L\kern-.36em%
{\sbox\z@ T%
\ vbox to\ht\z@{\hbox{%
\check@mathfonts
\fontsize\sf@size\z@
\math@fontsfalse\selectfont A}%
\vss}%
}%
\kern-.15em%
\TeX}
```

and mine

```
\DeclareRobustCommand{\LaTeX}{%
{%
L%
\setbox\z@\hbox{\check@mathfonts
\fontsize\sf@size\z@
\math@fontsfalse\selectfont
A}%
\kern-.57\wd\z@
\sbox\tw@ T%
\ vbox to\ht\tw@{\copy\z@ \vss}%
\kern-.2\wd\z@}%
}
```

```
{%
\ifdim\fontdimen1\font=\z@
\else
\count\z@=\fontdimen5\font
\multiply\count\z@ by 64\relax
\divide\count\z@ by\p@
\count\tw@=\fontdimen1\font
\multiply\count\tw@ by\count\z@
\divide\count\tw@ by 64\relax
\divide\count\tw@ by\tw@
\kern-\the\count\tw@ sp\relax
\fi}%
\TeX}
```

Here are a few examples. Enjoy.

lmr L^AT_EX vs. L^AT_EX & (L^A)T_EX

lmss L^AT_EX vs. L^AT_EX & (L^A)T_EX

lmr L^AT_EX vs. L^AT_EX & (L^A)T_EX

lmr L^AT_EX vs. L^AT_EX & (L^A)T_EX

qpl L^AT_EX vs. L^AT_EX & (L^A)T_EX

qpl L^AT_EX vs. L^AT_EX & (L^A)T_EX

qtm L^AT_EX vs. L^AT_EX & (L^A)T_EX

qtm L^AT_EX vs. L^AT_EX & (L^A)T_EX

qbk L^AT_EX vs. L^AT_EX & (L^A)T_EX

qbk L^AT_EX vs. L^AT_EX & (L^A)T_EX

qzc L^AT_EX vs. L^AT_EX & (L^A)T_EX

qhv L^AT_EX vs. L^AT_EX & (L^A)T_EX

qhv L^AT_EX vs. L^AT_EX & (L^A)T_EX

iwona L^AT_EX vs. L^AT_EX & (L^A)T_EX

iwona L^AT_EX vs. L^AT_EX & (L^A)T_EX

Benefits, care and feeding of the `bigfoot` package

David Kastrup
dak (at) gnu dot org

Abstract

The `bigfoot` package for arranging footnote apparatus for text-critical editions offers several advantages for ‘ordinary’ documents as well. The author plans to release a few enhancements in time for the EuroBachTeX conference which will further help in making it useful for other documents without having to think too much.

In the easiest case, just using `\usepackage{bigfoot}` in your preamble should provide for better page breaks and footnote arrangements.

There are, unfortunately, also some possible conflicts with other packages. The talk will focus on how to address them, and possibilities of still using `bigfoot` in such cases.

1 Advanced features of `bigfoot`

`bigfoot` has been designed to deal with the typesetting needs of a complicated critical edition. As a consequence, it offers multiple footnote apparatus. For doing that, it builds upon the interfaces and functionality of the `manyfoot` package. However, its functionality far exceeds that of `manyfoot`. It is, for example, also possible to anchor footnotes within any footnote apparatus previously on the page as well as in the main text (if the original author already used footnotes, not uncommon in critical editions from the last few centuries, comments on both his main text as well as his footnotes have to be permitted). In connection with the supporting package `perpage`, the numbering and order irregularities caused by being able to anchor footnotes in different other blocks get ironed out to get a natural page order.

Most talks about `bigfoot` have focused on demonstrating how `bigfoot` is able to deal pleasingly with the special demands of typesetting critical editions.

So what does `bigfoot` offer the average user? Let us first analyze what TeX does not offer.

2 The problems with TeX’s footnotes

Footnotes are one of TeX’s weakest points, and the principal weakness is breaking them. As soon as a footnote does not fit completely on one page, TeX’s global pagebreak optimization gets completely bypassed.

What TeX does upon encountering a footnote that will not fit on the current page is tentatively split it to fit in the remaining page size, using the standard `\vsplit` operation and registering the nat-

ural size to put on the current page. It then proceeds with the normal page accumulation and breaking.

There are so many things wrong with this approach that it is not easy to list them. The first thing wrong is that only one break of the footnote will be considered, though it may be more appropriate to break the footnote earlier and get more main text material instead. The worst aspect is that the split of the footnote is calculated before it is even clear that there will be a corresponding legal breakpoint in the main text!

If, for example, widows (page breaks before the last line of a paragraph) are not permitted by setting `\widowpenalty` to 10000, an action not uncommon in document classes, a footnote anchored in the second to last line of a paragraph will simply not get broken in normal circumstances, since the break of the footnote will be determined without taking into account that a line of the main text is still forced to follow.

Another problem is that the `\vsplit` operation takes into account any existing shrinkability in the top part of the split, thus possibly cramming more material into it than would ‘naturally’ fit. But since TeX considers only the natural height of the split part when it comes to page break decision time, it can happen that the split was chosen in a manner that lets TeX look at an overfull page. Again, this means that the footnote can’t be placed at all on the current page.

And we are not even talking about multiple footnotes yet ...

3 Features

So what are the features that `bigfoot` provides for the case of a ‘normal’, single apparatus?

Robustness `\verbatim` commands are allowed in footnotes. This is actually not as much a deficiency of `TEX`, but rather of the implementation in `LATEX`. Plain `TEX` has working functionality in this area.

The problem with `LATEX` lies in the footnote being scanned first as a macro argument. This is usually done by the typically document-class dependent `\@makefntext` command. The trickery `bigfoot` does here is too awful to describe, yet astonishingly works with most typical definitions of this macro. Where it doesn't, one can specify the `fragile` option to the `bigfoot` package, and this magic will not get used.

Optimization Footnote breakpoints are reconsidered for each possible breakpoint of the main text. This means that `TEX` will find the best combination of breaks in main text and footnote. In contrast, the default behavior examines just a single break possibility for a footnote, and this possibility might even be infinitely bad.

Color continuity When a footnote breaks across pages, the color stack is maintained properly. Color is handled in `LATEX` with the help of specials that switch the color (and, in the case of `dvips`, restore it afterwards with the help of a color stack). Restarting the footnote on the next page with the proper color is something that has never worked in `LATEX`. Now it simply does. It has to be noted that `pdfTEX 1.40`, the version in `TEX Live 2007`, has a built-in color stack feature that can be used to similar effect in PDF mode. It won't be likely to help in DVI mode, though.

Paragraph footnotes Footnotes may be set in a compact form in one running paragraph where this seems feasible. While `manyfoot` and `fnpara` also offer this arrangement, `bigfoot` offers a superior solution in several respects:

- The line breaking can be chosen much more flexibly: with appropriate customization, it is possible to fine-tune quite well when and where stuff will be placed in the same line, and when starting a new line will be preferred.
- Such in-paragraph footnotes can be broken across pages automatically, just like normal footnotes. They will only be broken after the last footnote in the block has started.
- Pages will not become over- or underfull because of misestimating of the size of in-paragraph footnotes.

- The decision of whether to make a footnote in-paragraph or standalone can be changed for each footnote apparatus at any time, including on mid-page. In fact, you can make this decision for each footnote separately. Since display math requires vertical mode footnotes, this is convenient.
- `bigfoot` will make a good-faith effort to adapt the normal footnote layout provided by the document class with the macros `\@makefnmark` and `\@makefntext` to in-paragraph footnotes.

Fewer catastrophes Split footnotes will not get jumbled in the presence of floats. `bigfoot` is not afflicted by this bug in `LATEX`'s output routine since it does not delegate the task of splitting footnotes to `TEX` in the first place. While the faulty output routine of `LATEX` may still jumble the order of footnotes in that particular case (when one footnote gets held over as an insertion 'floated' at infinite cost), `bigfoot` will sort the jumbled footnotes back into order before processing them.

However, it must be noted that the bug of a footnote getting detached from its anchor line when followed by a float anchored on the same line is still present: the marks that `bigfoot` employs instead of insertions for keeping track of the insertion positions can get detached in the same manner.

4 Drawbacks in practice

- Since `bigfoot` meddles considerably with the output routine's workings, interoperation with other packages doing the same might be problematic. Considerable effort has been spent on minimizing possibly bad interactions, but the results might not always be satisfactory and, at the very least, might depend on the load order of packages. So playing around with the load order might help.
- The underlying `manyfoot` changes some `LATEX` internals. Packages that do similar operations might clash. One such clash has very recently been addressed in `jurabib`.
- It slows things down. In practice, this is most noticeable for multiple apparatus where there are no good alternatives, anyway.
- The complexity of the package makes it more likely for things to go wrong in new ways.¹

¹ Most of those problems should arise under requirements that could not possibly be met without the package, so this

- The robustification of footnotes might not work with all document classes. It is worth trying to load the `bigfoot` package with the `fragile` package option. This has been made available only recently.
- The version distributed in T_EX Live 2007 can still get overfull pages and suboptimal breaks. A revision is underway and should be finished at the time of the conference.
- Documentation is sparse and not optimal.

would be reason for improving rather than not using the package.

5 Using it

Simply `\usepackage{bigfoot}` should work for the average case and improve page layout and breaks. If you want to have short footnotes possibly placed inside of a paragraph, use

```
\AtBeginDocument{%  
  \RestyleFootnote{default}{para}}
```

You will not likely notice much of a change at first, unless you actually use short footnotes. For long footnotes, paragraph mode is ungainly and thus avoided automatically.

MathPSfrag: L^AT_EX labels in MATHEMATICA plots

Johannes Große

Institute of Physics, Jagiellonian University, Reymonta 4, 30-059 Kraków, Poland

jgrosse (at) mppmu dot mpg dot de

<http://wwwth.mppmu.mpg.de/members/jgrosse>

Abstract

A MATHEMATICA[®] package that allows inclusion of L^AT_EX labels in EPS graphics using PSfrag will be presented. The clue is that positioning information and T_EX code is automatically generated by the package. It also contains a preview capability that imports a bitmap of the final image including the rendered L^AT_EX labels back into MATHEMATICA.

1 Introduction

MATHEMATICA (Wolfram, 1999; Wolfram Research, Inc., 2005) is one of the major commercial computer algebra systems and as such used in many fields of scientific research.

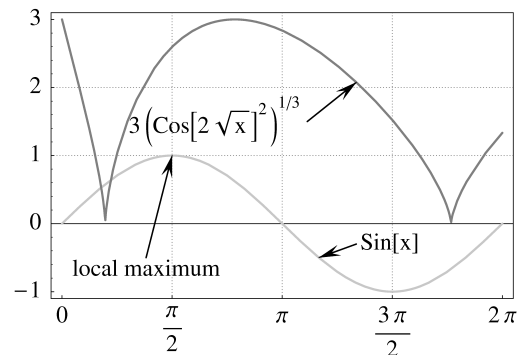
Unfortunately, labels in graphics produced by MATHEMATICA — like those of most other graphics programs — are not visually compatible with T_EX's standard fonts. Even though MATHEMATICA provides advanced typesetting capabilities for complex mathematical expressions that are close to a faithful representation of standard mathematical notation, it cannot compete with T_EX in this regard.

MathPSfrag (Große, 2005) is intended to fill this gap, but it is also meant to address another problem: Many authors consider the layout of the manuscript as something to be safely left to the computer. While T_EX does a remarkable job in providing excellent typesetting with little user intervention, the same cannot be said about image preparation.

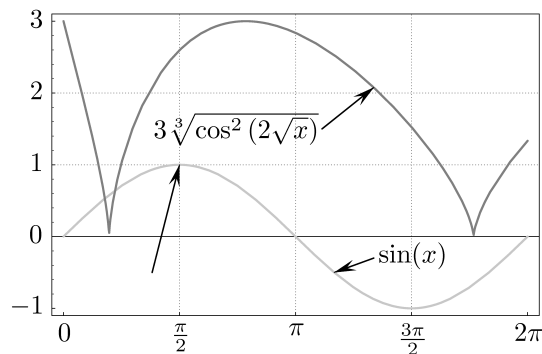
Solutions to the common task of attaching labels to plots range from sophisticated (McKay and Moore, 1999) to crude: conversion of the exported EPS file to JPEG, editing in a graphics program, back conversion to EPS for inclusion in L^AT_EX.

From the latter example it is clear that any solution addressing this problem needs to work not only from a typesetting point of view but also from a daily user's perspective. MathPSfrag is an attempt to combine the existing technique of PSfrag (Grant and Carlisle, 1998) with a transparent, easy to use convenience layer.

The PSfrag package provides T_EX macros that allow replacement of text strings (“tags”) in EPS graphics. For PSfrag to work these tags have to be output unaltered using a single PostScript `show` directive. Since MATHEMATICA splits complicated expressions into several `show` commands, simple alphanumeric



(a) Conventional MATHEMATICA without MathPSfrag



(b) The same plot after automatically substituting all Text primitives (including tick mark labels) by L^AT_EX output.

Figure 1: Old vs. new graphics export

sequences have to be used as tags, which makes the resulting raw EPS file rather illegible. Bookkeeping of automatically generated tags was the only feature provided by the very first version of MathPSfrag, although several more sophisticated features have been added since.

Ideally MathPSfrag does not require any user

```
f1[x_] := Sin[x]; f2[x_] := 3*((Cos[2 Sqrt[x]])^2)^(1/3);

rawplot = Plot[{f1[x], f2[x]}, {x, 0, 2 Pi}, PlotStyle→{Hue[1.0], Hue[0.6]}, Frame→True,
  FrameTicks→{Pi/2*{0, 1, 2, 3, 4}, Automatic, None, None}];

Needs["Graphics`Arrow`"];
SimpleLabel[tip : {_, _}, txt_, txtpos : {_, _}, align : {_, _}] := Sequence[
  Arrow[txtpos, tip, HeadScaling→Absolute, HeadLength→8, HeadCenter→0.6],
  Text[txt, txtpos, align]];
textlabels = Graphics[{
  SimpleLabel[{Pi/2, f1[Pi/2]}, "local maximum", {1, -0.5}, {0, 1}],
  SimpleLabel[{7/6Pi, f1[7/6Pi]}, f1[x], {4.2, -0.3}, {-1, 0}],
  SimpleLabel[{4.2, f2[4.2]}, f2[x], {3.5, 1.5}, {1, 0}]
}];
mygrid = Map[#, {AbsoluteDashing[{0.1, 1}], GrayLevel[0.5]}] &, {Pi*{1/2, 1, 3/2}, {1, 2}}, {2}];
exampleplot = Show[rawplot, textlabels, GridLines→mygrid];
```

Figure 2: Full MATHEMATICA code for the plot in Figure 1(a).

intervention except for calling a different graphics export command from within MATHEMATICA.

MathPSfrag will take over the task of inserting tags into the EPS in place of the original labels and will also use MATHEMATICA’s TeXForm command to determine the L^AT_EX macros reproducing a pretty-printed version of the original MATHEMATICA expression. These macros are written to a separate T_EX file. In a second step, a L^AT_EX, dvips, Ghostscript sequence is carried out to merge the two files to a single EPS that is independent of PSfrag and will be called “unpsfraged” in the following. As such it can be (and by default is) converted into a PDF image suitable for inclusion in pdfL^AT_EX documents. Moreover a bitmap image is rendered and imported back into MATHEMATICA as a preview.

In reality there are a number of problems that can arise—the simplest would be MATHEMATICA producing undesired or flawed T_EX code, such that the above rendering sequence would fail. Since the process of image creation described in this article involves many programs and production steps, there is actually quite a lot of potential for malfunction. In the first section of this tutorial we will nevertheless assume that this does not happen and that MathPSfrag has already been set up correctly to find L^AT_EX, dvips and Ghostscript. In subsequent sections, we will discuss these points in more detail. For a full presentation of all options and extended examples, the reader is referred to the manual accompanying MathPSfrag.

We would like to point out that we denote three different objects *psfrag*: the L^AT_EX package PSfrag, which provides the L^AT_EX macro \psfrag, and its MATHEMATICA counterpart PSfrag.

2 A first example

For concreteness we will start by defining a conventional MATHEMATICA plot without any reference to MathPSfrag. We will try to make it as beautiful as possible for a fair comparison with MathPSfrag. The code given in Figure 2 performs the following actions to draw Figure 1(a).

The first line defines the functions to be plotted, which happens in the second line. This already gives a decent plot, but to show off MathPSfrag’s capabilities a few more elements are inserted into the plot. The third block of commands loads a standard MATHEMATICA package and defines the function SimpleLabel, which draws an arrow and attaches a textual expression to the arrow. It is then used to define the three text labels seen in the plot. (By “textual expression” we denote all expressions that at some point end up as the argument of a Text primitive, in other words the expressions we want to replace by L^AT_EX commands eventually.) As a finishing touch, a grid of light gray lines is created. The last line merges all those elements into the final plots in Figure 1.

An EPS image can be produced by a simple Export[exampleplot, "exampleplot.eps"]. However, by default MATHEMATICA unfortunately uses Courier as a font for the labels, and does not allow inclusion of fonts into the EPS image (for MATHEMATICA versions before 4.2.1); as a result, any symbols, such as large brackets, that require MATHEMATICA’s special fonts can only be processed when the T_EX distribution has been set up to find these fonts (WRI Support, 2007). For later MATHEMATICA versions we should rather export the plot by:

```
Export[Show[exampleplot,
  TextStyle→{FontFamily→"Times"}],
  "pure-mma.eps", ConversionOptions→
  {"IncludeSpecialFonts"→True}]
```

which sets the default font to Times Roman.

The corresponding export commands provided by `MathPSfrag` read

```
Needs["MathPSfrag"];
PSfragExport[exampleplot, "filename"]//UnPSfrag;
```

which loads the `MathPSfrag` package and creates files `filename-psfrag.tex` and `filename-psfrag.eps` containing the `PSfrag` versions of the plot.

The `UnPSfrag` command takes these files and creates an unpsfraged PDF and EPS version, which in turn is rendered into a bitmap, imported into `MATHEMATICA` and displayed as a preview. When the user wants to use only the `PSfrag` versions of the images, the `UnPSfrag` command can be omitted. The package is accompanied by a shell script for merging the two files into an unpsfraged EPS file. This may be useful when there is no `LATEX` distribution available on the machine where `MATHEMATICA` resides.

In general, it is recommended *not to rescale* unpsfraged images within the `LATEX` source of the final manuscript, because such a rescaling would also change the size of the rendered `LATEX` expressions. While on modern `TEX` installations chances are good to end up with the scalable outline versions of Computer Modern in the EPS files because `MathPSfrag` by default invokes `dvips` with the `-Ppdf` switch, the overall visual consistency of the final manuscript will suffer from labels of different sizes. Hence it is recommended to set the size of the image at rendering time by providing suitable options to the `\includegraphics` command that is internally invoked by `UnPSfrag`. This can be achieved by

```
UnPSfrag[PSfragExport[exampleplot, "filename"],
  IncludeGraphicsOptions→"width=75mm"];
```

which will preserve the labels, while scaling the image *approximately* to the given size. The reason for the mismatch of size is that the bounding box of the final image correctly fits its *contents* while the size provided by the user refers to the bounding box of the original image, which changes during the `PSfrag` process.

When special symbols or different fonts are required for the graphics, `UnPSfrag` can be instructed to include a user-defined preamble by means of the `TeXPreamble` option.

3 Providing custom `LATEX` commands

`MathPSfrag` generates `LATEX` commands by employing `MATHEMATICA`'s `TeXForm`, whose output may not always be what the author expected. In particular, the output of `TeXForm` depends very much on the version of `MATHEMATICA`—versions 5.1 or later are most suitable, though a compatibility `TEX` package has been implemented; see ‘Known limitations’.

For overriding `MathPSfrag`'s default treatment of single textual elements of the plot, the `PSfrag` command is provided. It can be simply wrapped around the argument of a `Text` primitive or a plot option that eventually produces a `Text` primitive. Most frequently used examples for the latter case are the `PlotLabel` and `AxesLabel` options. So instead of `PlotLabel→"chi-square test"`, the following could be used:

```
PlotLabel→PSfrag["chi-square-test",
  TeXCommand→"$\\chi^2$-test"]
```

This would still display as “chi-square test” in `MATHEMATICA`, but appear as “ χ^2 -test” in the final manuscript. Note that a doubling of any backslash in the argument of the `TeXCommand` options is required because `MATHEMATICA` considers the backslash character to be an escape symbol.

Since `MATHEMATICA` provides the means for entering formatted expressions as part of ordinary text strings, the above example is somewhat artificial. The same effect could have been achieved by simply using `PlotLabel→"χ2-test"` and relying on `MathPSfrag` (or to be more precise `TeXForm`) to produce the corresponding `TEX` representation.

A more realistic example would be changing one of the labels in Figure 1(b) by replacing

$$3\sqrt[3]{\cos^2(2\sqrt{x})} \quad \text{by} \quad 3\left|\cos\sqrt{4x}\right|^{\frac{2}{3}}.$$

This can be achieved by substituting the argument of the corresponding `SimpleLabel` line in Figure 2.

```
tex="$3\\left|\\cos\\sqrt{4x}\\right|^{\frac{2}{3}}$";
...
SimpleLabel[{4.2, f2[4.2]},
  PSfrag[f2[x], TeXCommand→tex],
  {3.5, 1.5}, {1, 0}]
```

The additional command has been written in italics; the resulting plot is shown in Figure 3.

`PSfrag` can also be used to pass alignment information, (angular) orientation or a scale factor to `MathPSfrag`. The respective options (`TeXPosition`, `PSPosition`, `PSRotation`, `PSScaling`) are in a one-to-one correspondence with the options of the command `\psfrag` provided by the `LATEX` package. In

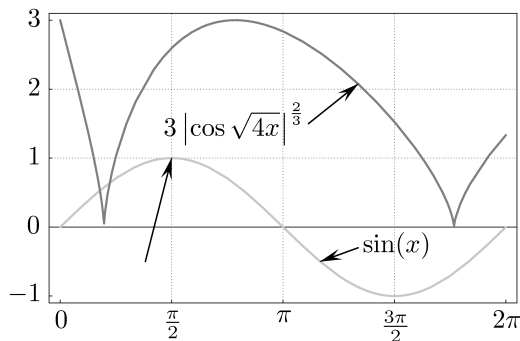


Figure 3: Manual replacement of the “cos...” label.

particular, the first two options accept strings built from two characters (top, center, **B**aseline, **b**ottom, plus left, right, center) describing the vertical and horizontal position of an anchor point in the tag and L^AT_EX box. When replacing the tag by the L^AT_EX box, the new anchor point is glued to the position of the old one. Rotation is in degrees; the use of the scale factor is discouraged and provided only for completeness’ sake.

Unless three-dimensional plots are used none of the above should be necessary. As a last resort, when there is a positioning bug, one may use the `TeXShift→{"x","y"}` option. `MathPSfrag` shifts the content of the corresponding expression by the amount specified in the two strings, which should contain valid T_EX dimensions.

4 Setup

As mentioned in the introduction, we left out some crucial points. The most important of these is that `MathPSfrag` needs information about the actual location of the L^AT_EX, `dvips` and `Ghostscript` executables unless they can be found in the system’s execution path. (Specifically, these binaries are needed by `UnPSfrag` whereas the other parts of `MathPSfrag` will continue to work without them. This is the main reason for having a separate `UnPSfrag` command at all.) While this is usually the case for Unix-like operating systems, it is rather the exception for Windows-based systems. Moreover, `MathPSfrag` by default uses the typical system-specific names of the executables, which might differ on some installations.

The user can either fix the system settings or provide the absolute path to the executables by setting the appropriate `UnPSfrag` options as outlined in Figure 4(a). The configuration may be checked by executing `MathPSfragConfigurationTest`, which will output diagnostic information. Step-by-step instructions guiding through the configuration are provided in a separate MATHEMATICA notebook

(“`MathPSfrag-Test.nb`”), which also generates a number of examples.

In order to avoid the necessity of setting the correct paths each time `MathPSfrag` is loaded, the configuration can also be stored in an `init.m` file, which is automatically loaded by MATHEMATICA during start-up. Valid locations of the `init.m` file depend both on the operating system and on the MATHEMATICA version, but are documented in MATHEMATICA’s Help Browser. A typical example for such a file is given in Figure 4(b).

```
SetOptions[UnPSfrag,
  LaTeXExecutable→"C:\\path-to\\latex.exe",
  DvipsExecutable→"C:\\path-to\\dvips.exe",
  GhostscriptExecutable→
    "C:\\path-to\\gswin32c.exe"];
```

(a) Setting the locations of external programs

```
AppendTo[$Path, "C:\\path-to\\MathPSfrag"];
$PostMathPSfrag := SetOptions[UnPSfrag,...];
```

(b) Minimal `init.m` file

Figure 4: Configuration of `MathPSfrag`

5 In the manuscript

Unpsfraged graphics can be treated in the usual manner and included by the `\includegraphics` command, where it is good practice to leave out the file’s suffix to allow L^AT_EX or pdfL^AT_EX to load the appropriate format. For best quality it is recommended to avoid usage of the `width` or `height` options, but instead to set the size of the plot during creation from within MATHEMATICA as described in ‘A first example’.

PSfrag-based graphics are generically included as follows:

```
\usepackage{psfrag,graphicx}
...
\begin{psfrags}
\input{exampleplot-psfrag}
\includegraphics[width=75mm,
  trim=-10 -20 -30 -40]{exampleplot-psfrag}
\end{psfrags}
```

where any numbers of course have to be adapted. The `trim` option enlarges the bounding box when negative arguments are used, which is sometimes required to avoid clipping problems. Hence one should always check the bounding box by enclosing each `\includegraphics` macro by an `\fbox`.

6 Known limitations

As mentioned in the introduction for the sake of presentation we have ignored all potential problems so far. It is however vital to point out that image production with `MathPSfrag` employs several programs which come in different versions and installations on different computer systems with all the associated compatibility problems. The work flow also involves a number of user decisions, which have a strong impact on the final image. For the discussion of these choices it is convenient to think of the process of manuscript creation in terms of the following stages.

1. Plot preparation with `MATHEMATICA`
2. Export with `MathPSfrag`
3. Manuscript preparation (and inclusion of images)
4. Output format generation (PS or PDF)

We will discuss these stages in reverse order. Let us assume that the final output format should be PDF. There are currently two possibilities to achieve this. Either following the traditional path of translating the manuscript with `LATEX`, `dvips` and a distillation to PDF, e.g., by `ps2pdf`; or using `pdfLATEX` with its enhanced typesetting capabilities. Since this choice is not necessarily under the author's control, it may be wise to keep both paths open.

When using unpsfraged images, this amounts to simply invoking `\includegraphics` without providing the filename's suffix and placing both the EPS and PDF version of the image where `TEX` can find them. When using `PSfrag`-based images, due to their PostScript-centric nature, additional effort is necessary to make these work with `pdfLATEX`. Fortunately, because of the popularity of the `pstricks` package there are several packages that provide methods for incorporating PostScript into `pdfLATEX` documents, namely `pdftricks`, `pst-pdf` and `ps4pdf`. While they differ considerably regarding ease-of-use and limitations of the respective implementation, all of them generate PDF versions of PostScript related images essentially by extracting them from a conventional `LATEX` run. An example file for each of those packages accompanies `MathPSfrag`.

Before deciding whether to employ `PSfrag`-based or unpsfraged images, one should keep in mind that unpsfraged images are hard to edit: They neither allow rescaling without changing the size of the labels nor is there an easy method of changing the contents of the labels. It is therefore advisable to design the `MATHEMATICA` notebook generating the plots in such a way that replotting can be achieved without recalculating. In other words the result of a time-consuming calculation should be stored sepa-

rately before plotting. Moreover the author should know in advance which fonts will be used for the final manuscript. These can be loaded by setting the `TeXPreamble` option of `UnPSfrag` to a suitable `\usepackage` command.

For `PSfrag`-based images changing a label only requires editing the corresponding `\psfrag` macro in the `TEX` file associated to the image, whereas rescaling of the image will preserve the size of the labels. However, `PSfrag`-based images always have a wrong bounding box, which can potentially lead to clipping errors and should therefore be manually corrected by use of the `trim` option for `\includegraphics`. Wrapping an `\fbox` around the image while doing so considerably facilitates this task.

The limitations of `MathPSfrag` we now discuss are mainly due to its dependence on three `MATHEMATICA` functions: `TeXForm`, `FullGraphics` and `AbsoluteOptions`.

The output of `TeXForm` consists of a `MATHEMATICA`-specific set of `LATEX` commands for versions earlier than 5.1, whereas starting from 5.1 `amsmath` macros are used. While still having deficiencies regarding symbols that do not have a direct `LATEX` counterpart, the latter is most suitable for use with `MathPSfrag`. The output of earlier `MATHEMATICA` versions, on the other hand, except for very basic expressions, will require a compatibility layer, which is part of the `MathPSfrag` distribution. It does however need to be installed where `LATEX` can find it when called by `UnPSfrag` and should also accompany the manuscript when `PSfrag`-based images are used. Alternatively it is of course possible to manually provide `LATEX` macros with the `PSfrag` command as described in 'Providing custom `LATEX` commands'.

Both `FullGraphics` and `AbsoluteOptions` convert (certain aspects of) `MATHEMATICA` graphics from a logical to a physical representation in terms of so-called primitives. `MathPSfrag` needs this physical representation for the extraction of alignment information of all text elements of a plot as well as for content generation in the case of tick marks. Unfortunately, neither command is faithful; i.e., they do not preserve the visual appearance of the plot. `MathPSfrag` has been carefully implemented to work around these shortcomings, but fails at one minor point: Floating point numbers very close to integers (difference $< 10^{-10}$) will be rounded.

While `MathPSfrag` should be able to correctly position and align any text elements, the respective options of `PSfrag` can also be used as a quick and dirty solution to any misplacements. In particular the `TeXShift` option is provided solely for this purpose since it is not used during default processing

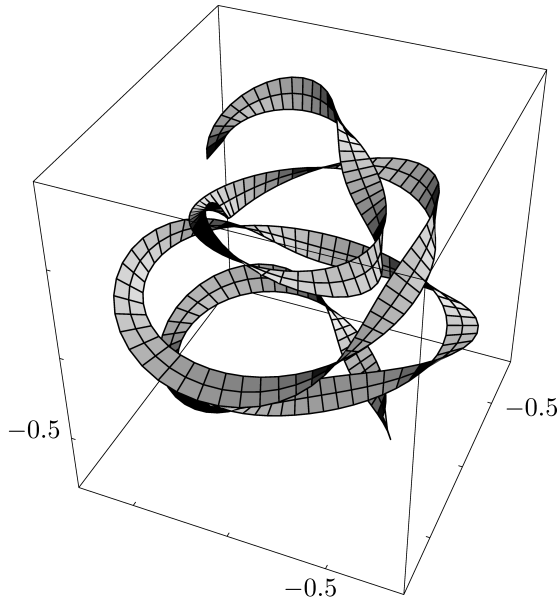


Figure 5: Three dimensional example: As there exists no `FullGraphics3D` command, manual labeling was required.

at all. In the case of any such misplacements, the author would appreciate a bug report.

Moreover there is no `FullGraphics` command for three-dimensional plots. As a consequence, the required alignment information has to be provided by hand for every text element of a plot. This sounds more tedious than it actually is in most cases — example code is provided as part of `MathPSfrag`. Here it shall suffice to show the result; cf. Figure 5.

Finally, `MathPSfrag` does not provide methods to construct correct tick mark *content* as it is strictly focused on shape. It does however integrate well with the `CustomTicks` package (Caprio, 2005), which provides that functionality.

7 Conclusion

`MathPSfrag` provides a convenient interface to `PSfrag` permitting the generation of high-quality labels in `MATHEMATICA` graphics. While it automates all tedious aspects of `PSfrag`, it still allows the user to seamlessly override all of its internal assumptions. The possibility to create images that do not depend on `PSfrag` anymore provides a simple method for achieving pdfL^AT_EX compatibility.

`MathPSfrag` has been tested with `MATHEMATICA` versions 4.1, 5.0 and 5.2 under Linux and Windows XP. A previous version has also been tested under Mac OS X.

For the future it would be interesting to incorporate the `PSfragx` extension that allows including the `\psfrag` commands into the comment lines of the EPS file. It would also be interesting to provide means for the generation of `\overpic` commands, thus bypassing many of the shortcomings of the `PSfrag` approach. For `MATHEMATICA`, because of its closed source nature, this is not a simple task because the position information of graphics primitives in terms of absolute coordinates for the final image is not easily accessible.

8 Acknowledgments

The author acknowledges support by ENRAGE (European Network on Random Geometry), a Marie Curie Research Training Network in the European Community’s Sixth Framework Programme, network contract MRTN-CT-2004-005616. Part of the work on `MathPSfrag` were supported by the DFG Graduiertenkolleg “The Standard Model of Particle Physics — structure, precision tests and extensions” at Humboldt-Universität zu Berlin and the Max-Planck-Institut für Physik, München.

References

- Caprio, Mark. “Custom tick marks for linear, logarithmic, and general nonlinear axes”. <http://library.wolfram.com/infocenter/MathSource/5599/>, 2005.
- Grant, Micheal C., and D. Carlisle. “The PSfrag system, version 3”. Available from CTAN, `macros/latex/contrib/psfrag`, 1998.
- Große, Johannes. “MathPSfrag”. <http://wwwth.mppmu.mpg.de/members/jgrosse/mathpsfrag>, 2005.
- McKay, Wendy, and R. Moore. “Convenient Labelling of Graphics, the WARMreader Way”. *TUGboat* **20**(3), 262–271, 1999. <http://www.tug.org/TUGboat/Articles/tb20-3/tb64ross.pdf>.
- Wolfram, Stephen. *The Mathematica book*. Cambridge University Press, New York, NY, USA, 4th edition, 1999.
- Wolfram Research, Inc. “Mathematica 5.2”. 2005. <http://www.wolfram.com/>.
- WRI Support. “MATHEMATICA Fonts in EPS files and Ghostscript”. Available from <http://support.wolfram.com/mathematica/graphics/export/ghostscript.html>, `includefonts.html`, 2007.

makematch, a L^AT_EX package for pattern matching with wildcards

David Kastrup

David dot Kastrup (at) QuinScape dot de

Abstract

The `makematch` package has been factored out from the `qstest` package and has the purpose of matching patterns with wildcards against targets. There is a generalization provided for matching (ordered) pattern lists against (unordered) target lists, in which case one can use commas or other separators (including spaces) for separating the list elements. The wildcard `*` matches zero or more arbitrary characters. Prepending `!` to a pattern will cause a match of it to revert possible matches from earlier in the pattern list.

Matching, for example, the pattern list

```
test*, !test10*b, !fails
```

with the target list

```
fails, test20
```

will lead to a non-match: while `test20` is matched by the pattern `test*`, the additional matching pattern `!fails` later in the list reverts this match.

Both pattern and target lists get ‘sanitized’ (converted into a unique printing form where no T_EX characters are interpreted specially) and compiled into a form which makes the matching itself quite efficient.

1 Using makematch

The basic idea of `makematch` is to compile patterns and targets (and/or lists of them) and match the former to the latter. This functionality is used extensively in QuinScape’s `qstest` package for unit testing. We’ll use that package for documenting usage of `makematch`; the following construct skips the tests when `makematch.dtx` is used as a standalone file.

```
<*dtx>
\iffalse
</dtx>
<*test>
\RequirePackage{makematch,qstest}
\IncludeTests{*}
```

`makematch` requires L^AT_EX to be based on ϵ -T_EX, which should be standard for current T_EX distributions.

1.1 Match patterns and targets

This package has the notion of match patterns and targets. Patterns and targets get sanitized at the time they are specified; this means that nothing gets expanded, but replaced by a textual representation consisting of spaces (with catcode 10) and other characters (catcode 12). Control words are usually followed by a single space when sanitized.

Patterns and targets are actually generalized to pattern and target lists by this package: you can, when specifying either, instead give a list by using an optional argument for specifying a list separator (the lists used in `qstest` are comma-separated).

Target lists are unordered: the order of targets in them is irrelevant. Leading spaces in front of each target get stripped; all others are retained.

Pattern lists similarly consist of a list of patterns, with leading spaces stripped from each pattern. In contrast to target lists, the order of pattern lists is significant, with later patterns overriding earlier ones. Also in contrast to target lists, empty patterns are removed.

There are two special characters inside of a pattern: the first is the wildcard `*` which matches any number of consecutive characters (including the empty string) in a target. Wildcards can occur anywhere and more than once in a pattern.

The second special character in a pattern is only recognized at the beginning of a pattern, and only if that pattern is part of a pattern list (namely, when a list separator is specified).¹ If a pattern is preceded by `!` then the following pattern, if it matches, causes any previous match from the pattern list to be disregarded.

¹ And if `!` is not the list separator of the list.

So for example, the pattern list `{*,!foo}` matches any target list that does not contain the match target `foo`.

An empty target list `{}` is considered to contain the empty string. Thus the pattern `*` matches every target list, including empty ones, while the pattern list `{}` does not match any keyword list, including empty ones.

1.2 The interface

The `\MakeMatcher` command takes two mandatory arguments. The first is a macro name. This macro will become the new matcher. The second argument of `\MakeMatcher` is the pattern to match. An optional argument before the mandatory ones can be used for specifying a list separator, in which case the first mandatory argument becomes a pattern list (only in this case are leading `!` characters before list elements interpreted specially).

```
\begin{qstest}{\MakeMatcher}{\MakeMatcher}
\MakeMatcher\stylefiles{*.sty}
\MakeMatcher\headbang{!*}
\MakeMatcher[,]\truestylefiles
{*.sty,! .thumbnails/*, !*/.thumbnails/*}
```

The matcher constructed in this manner is called with three arguments. The first argument is a control sequence name containing a match target (or target list) prepared using `\MakeMatchTarget` (see below).

Alternatively, the first argument can be a brace-enclosed list (note that you'll need *two* nested levels of braces, one for enclosing the argument, one for specifying that this is a list) which will then get passed to `\MakeMatchTarget` (see below) for processing before use. The inner level of braces inside of the first argument may be preceded by a bracketed optional argument specifying the list separator for this list.

The second argument of the matcher is executed if the pattern list for which the matcher has been built matches the keyword list. The third is executed if it doesn't. List separators of pattern and keyword list are completely independent from each other. So, we expect the following to result just in calls of `\true` (a call of `\false` is turned into a failed expectation):

```
\begin{qstest}{\Makematcher literal}
\MakeMatcher}
\begin{ExpectCallSequence}
{\true}{\false}{%
'.#1{\Expect*{\CalledName#1}{true}}+'}
\stylefiles
{xxx/.thumbnails/blubb.sty}}
{\true}{\false}
```

```
\truestylefiles
{xxx/.thumbnails/blubb.sty}}
{\false}{\true}
\headbang
{xxx/.thumbnails/blubb.sty}}
{\false}{\true}
\stylefiles
{[ ]{x.sty.gz .thumbnails/x.sty !x}}
{\true}{\false}
\truestylefiles
{[ ]{x.sty.gz .thumbnails/x.sty !x}}
{\false}{\true}
\headbang
{[ ]{x.sty.gz .thumbnails/x.sty !x}}
{\true}{\false}
\end{ExpectCallSequence}
\end{qstest}
```

So how do we create a sanitized keyword list in a control sequence?

`\MakeMatchTarget` is called with two mandatory arguments, the first being a control sequence name where the keyword list in the second argument will get stored in a sanitized form: it is converted without expansion to characters of either “other” or “space” category (catcodes 12 and 10, respectively), and any leading spaces at the beginning of an element are removed. Without an optional bracketed argument, nothing more than sanitization and leading space stripping is done. If an optional bracketed argument before the mandatory arguments is specified, it defines the list separator: this has to be a single sanitized character token (either a space or a character of category “other”) that is used as the list separator for the input (the finished list will actually always use the macro `\`, as a list separator).

```
\begin{qstest}{\Makematcher%
&\MakeMatchTarget}%
{\MakeMatcher,%
\MakeMatchTarget}
\MakeMatchTarget\single
{xxx/.thumbnails/blubb.sty}
\MakeMatchTarget[ ]
\multiple{x.sty.gz
 .thumbnails/x.sty !x}
\begin{ExpectCallSequence}
{\true}{\false}{%
'.#1{\Expect*{\CalledName#1}
{true}}+'}
\stylefiles{\single}{\true}{\false}
\truestylefiles\single{\false}{\true}
\headbang\single{\false}{\true}
\stylefiles{\multiple}{\true}{\false}
\truestylefiles\multiple{\false}{\true}
\headbang\multiple{\true}{\false}
\end{ExpectCallSequence}
\end{qstest}
```

After a match process `\MatchedTarget` will contain the target matched by the last matching pattern (if several targets in a match target list match, only the first of those is considered and recorded), regardless of whether the corresponding pattern was negated with `!`. After a successful match, you can call `\RemoveMatched` with one argument: the control sequence name where the list was kept, and the match will get removed from the list. If every list element is removed, the list will be identical to `\@empty`.

```

\begin{qstest}{\MatchedTarget}
    {\MakeMatcher,%
     \MakeMatchTarget,%
     \MatchedTarget}
\MakeMatchTarget\single
  {xxx/.thumbnails/blubb.sty}
\MakeMatchTarget[\ ]\multiple
  {x.sty.gz .thumbnails/x.sty !x}
\begin{ExpectCallSequence}
  {\true}{\false}{%
   '.#1{\Expect*{\CalledName#1}
    {true}}+'}
\stylefiles{\single}
  {\true}{\false}
\Expect*{\single}
  {xxx/.thumbnails/blubb.sty}
\Expect*{\meaning\MatchedTarget}
  *{\meaning\single}
\RemoveMatched\single
\Expect*{\meaning\single}
  {\macro:->}
\trstylefiles\single
  {\false}{\true}
\headbang\single
  {\false}{\true}
\stylefiles{\multiple}
  {\true}{\false}
\Expect*{\MatchedTarget}
  {.thumbnails/x.sty}
\RemoveMatched\multiple
\Expect[expandafter]{\multiple}
  {x.sty.gz\,!x}
\trstylefiles\multiple
  {\false}{\true}
\Expect*{\meaning\MatchedTarget}
  {undefined}
\headbang\multiple
  {\true}{\false}
\Expect*{\MatchedTarget}{!x}
\RemoveMatched\multiple
\Expect*{\multiple}{x.sty.gz}
\end{ExpectCallSequence}
\end{qstest}
\end{qstest}

```

1.3 Notes on sanitization

Note that sanitization to printable characters has several consequences: it means that the control sequence `\`, will turn into the string `\` followed by the end of the keyword. Note also that single-character control sequences with a nonletter name are not followed by a space in sanitization. This means that sanitization depends on the current catcodes. Most particularly, sanitizing the input `\@abc12` will turn into `\@abc 12` when `@` is of catcode letter, but to `\@abc12` when `@` is a nonletter.

So sanitization cannot hide all effects of catcode differences. It is still essential since otherwise braces would cause rather severe complications during matching.

Another curiosity of sanitization is that explicit macro parameter characters (usually `#`) get duplicated while being sanitized.

This is the end of the documentation section, so we end our test file setup by complementing the beginning:

```

  {/test}
  {*dtx}
  {fi}
  {/dtx}

```

2 Conclusions and outlook

`makematch` sets out to solve the task of pattern matching with wildcards in a very efficient manner. One basic restriction for some applications might be that it is restricted to comparing sanitized token lists. This has the effect that it is not possible to hide material from matching by enclosing it in braces. On the other hand, `TEX` will strip enclosing braces around a matched argument, making it unreliable to repeat matches or what to expect from a matched string.

In a later version, possibly starred forms of the commands will be provided that omit the sanitization. Those will not be able to match several characters with a meaning particular to `TEX` (such as `#`, `{` or `}`), but will probably come handy in other situations, like parsing keyword lists yielding `TEX` arguments. While it is possible to do this with the current code, using `\scantokens` for turning them active again, this can cause matches leading to unpaired braces, and it will not make it possible to hide commata from the matching by enclosing them in braces.

qstest, a L^AT_EX package for unit tests

David Kastrup

David dot Kastrup (at) QuinScape dot de

Abstract

The `qstest` package was created because of the need to verify in a large L^AT_EX project that no regressions occur. The test environments ensure that macros and registers will be set to expected values in test code, and that macro calls occur in a certain sequence and with certain values. Tests are usually embedded directly into the source code of `.dtx` files, thus providing documentation as well as verification.

It is also possible to compare results of one test run to those of previous runs.

Several log files may be created simultaneously in order to record the results of tests ordered into various categories.

1 Using `qstest`

The basic idea of `qstest` is to let the user specify a number of tests that can be performed either at package load time or while running a separate test file through L^AT_EX. If you are writing `.dtx` files, it is a good idea to use `docstrip` ‘modules’ for specifying which lines are to be used for testing. The file `qstest.dtx` from which both the style file as well as this documentation have been generated has been written in this manner.

Since the tests should be ignored when the `dtx` file is itself compiled, we use this for skipping over the tests:

```
<*dtx>
\iffalse
</dtx>
```

A standalone test file does not need a preamble. We can load the packages with `\RequirePackage` and just go ahead. Let us demonstrate how to build such a test file by testing the `qstest` package itself:

```
<*test>
\RequirePackage{qstest}
```

1.1 Pattern and keyword lists

See the section “Match patterns and targets” of the `makematch` package for an explanation of the comma-separated pattern and keyword lists. In a nutshell, both are lists of arbitrary material that is not expanded but rather used in sanitized (printable) form. Patterns may contain wildcard characters `*` matching zero or more characters, and may be preceded by `!` in order to negate a match from an earlier pattern in the pattern list. Leading spaces before an item in either list are discarded.

1.2 Specifying test sets

`\IncludeTests` specifies a pattern list matched to tests’ keyword lists in order to determine the tests to be included in this test run. The characters `*` and `!` are interpreted as explained above.

For example,

```
\IncludeTests{*, !\cs}
```

will run all tests except those that have a test keyword of `\cs` in their list of keywords. It is a good convention to specify the principal macro or environment to be tested as the first keyword.

The default is to include all tests. If you are interspersing possibly expensive tests with your source file, you might want to specify something like

```
\IncludeTests{*, !expensive}
```

or even

```
\IncludeTests{}
```

in your document preamble, and then possibly override this on the command line with

```
latex "\AtBeginDocument{
\IncludeTests{*}}\input{file}"
```

or similar for getting a more complete test.

`\TestErrors` defines test patterns that will throw an error when failing. A test that throws an error will not also add a warning to the standard log file with extension `log` since that would be redundant.

The default is `\TestErrors{*, !fails}`, to have all tests that are not marked as broken throw an error when they fail.

The throwing of errors does not depend on the logging settings (see below) for the default `log` file.

`\LogTests` receives three arguments. The first is the filename extension of a log file (the extension

`log` is treated specially and uses package warning and info commands to log test failures and passes, respectively). The second is a keyword list that indicates which passed tests are to be logged. The third is a keyword list specifying which failed tests are to be logged. Let us open a file logging everything:

```
\LogTests{lgout}{*}{*}
```

The choice of `lgout` is made to make it possible to also have `lgin` for comparison purposes: the latter would be an `lgout` file from a previous, ‘definitive run’, renamed and checked into version control, for the sake of being able to compare the log output from different versions.

An already open log file stays open and just changes what is logged. By default, the standard log (pseudo-)file is already open and logs everything.

Passed and failed tests are not completely symmetric with regard to logging: test failures are logged and/or indicated on the individual failed assertions, while a successful test is only logged and/or indicated in summary.

With `\LogClose` you can explicitly close a log file if you want to reread it in the course of processing, or call an executable that would process it. The standard file with extension `log` will not actually get closed and flushed if you do this (though logging would stop on it), but all others might. An actual example for this follows after the tests. You can reopen a closed log file using `\LogTests`. It will then get rewritten from the beginning (with the exception of the standard `log` file, of course).

1.3 The tests

Tests are performed within the `qstest` environment. The environment gets two arguments. The first is the name of the test recorded in the log file. The second is a list of test keywords used for deciding which tests are performed and logged.

Before delving into the details of what kind of tests you can perform in this environment, we list the various commands that are given patterns and thus control what kind of tests are performed and logged.

`\Expect` is the workhorse for checking that values, dimensions, macros and other things are just what the test designer would expect them to be.

This macro basically receives two brace-delimited arguments¹ and checks that they are equal af-

¹ The arguments are collected with a token register assignment. This gives several options for specifying them, including giving a token register without braces around it. It also makes it possible to precede the *balanced text* with `\expandafter` and similar expandable stuff.

ter being passed through `\def` and sanitized. This means that you can’t normally use `#` except when followed by a digit (all from 1 to 9 are allowed) or `#`. If you precede one of those arguments with `*` it gets passed through `\edef` instead of `\def`. There may also be additional tokens like `\expandafter` before the opening brace. Note that the combination of `\edef` and `\the(token variable)` can be used to pass through `#` characters without interpretation. ε -TeX provides a similar effect with `\unexpanded`. So if you want to compare a token list that may contain isolated hash characters, you can do so by writing something like

```
(*etex)
\begin{qstest}{# in isolation}
    {\Expect, #, \unexpanded}
    \toks0{# and #}
    \Expect*{\the\toks0}
        *{\unexpanded{# and #}}
\end{qstest}
</etex>
```

Since the sanitized version will convert `#` macro parameters to the string `##`, you might also do this explicitly (and without ε -TeX) as

```
\begin{qstest}{# in isolation 2}
    {\Expect, #, \string}
    \toks0{# and #}
    \Expect*{\the\toks0}
        *{\string#\string#
          and \string#\string#}
\end{qstest}
```

If the token register is guaranteed to contain only ‘proper’ `#` characters that are followed by either another `#` or a digit, you can let the normal interpretation of a macro parameter for `\def` kick in and use this as

```
\begin{qstest}{# as macro parameter}
    {\Expect, #}
    \toks0{\def\xxx#1{}}
    \Expect\expandafter{\the\toks0}
        {\def\xxx#1{}}
\end{qstest}
```

In this manner, `#1` is interpreted (and sanitized) as a macro parameter on both sides, and consequently no doubling of `#` occurs.

Before the comparison is done, both arguments are sanitized, converted into printing characters with standardized catcodes throughout.² A word of warning: both sanitization as well as using `\meaning` still depend on catcode settings, since single-letter control sequences (made from a catcode 11 letter) are followed by a space, and other single-character control sequences are not. For this reason,

² Spaces get catcode 10, all other characters catcode 12.

a standalone test file for L^AT_EX class or package files will usually need to declare

```
\makeatletter
```

in order to make ‘@’ a letter, as is usual in such files.

All of the following expectations would turn out correct:

```
\begin{qstest}{Some LaTeX definitions}
  {\Expect}
  \Expect*{\meaning\@gobble}
    {\long macro:#1->}
  \Expect*{\the\maxdimen}
    {16383.99998pt}
\end{qstest}
```

Note that there is no way to convert the contents of a box into a printable rendition, so with regard to boxes, you will mostly be reduced to checking that the box dimensions meet expectations.

1.4 Expecting ifthen conditions

`\ExpectIfThen` is used for evaluating a condition as provided by the `ifthen` package. See its docs for the kind of condition that is possible there. You just specify one argument: the condition that you expect to be true. Here is an example:

```
\RequirePackage{ifthen}
\begin{qstest}{\ExpectIfThen}
  {\ExpectIfThen}
  \ExpectIfThen{
    \lengthtest
    {\maxdimen=16383.99998pt}\AND
    \maxdimen>1000000000}
\end{qstest}
```

1.5 Dimension ranges

`\InRange` checks not whether some dimension is exactly equal to some value, but rather within some range. We do this by specifying as the second argument to `\Expect` an artificial macro with two arguments specifying the range in question. This will make `\Expect` succeed if its first argument is in the range specified by the two arguments to `\InRange`.

The range is specified as two T_EX dimens. If you use a `dimen` register and you want to have a possible error message display the value instead of the `dimen` register, you can do so by using the `*` modifier before `\InRange` (which will cause the value to be expanded before printing and comparing) and put `\the` before the `dimen` register since such registers are not expandable by themselves.

Here are some examples:

```
\begin{qstest}{\InRange success}
  {\InRange}
  \dimen@=10pt
  \Expect*{\the\dimen@}
```

```
\InRange{5pt}{15pt}
\Expect*{\the\dimen@}
  \InRange{10pt}{15pt}
\Expect*{\the\dimen@}
  \InRange{5pt}{10pt}
\end{qstest}
\begin{qstest}{\InRange failure}
  {\InRange, fails}
  \dimen@=10pt \dimen@ii=9.99998pt
  \Expect*{\the\dimen@}
    \InRange{5pt}{\dimen@ii}
  \dimen@ii=10.00002pt
  \Expect*{\the\dimen@}
    *\InRange{\the\dimen@ii}{15pt}
\end{qstest}
```

`\NearTo` requires ϵ -T_EX’s arithmetic and so will not be available for versions built without ϵ -T_EX support. The macro is used in lieu of an expected value and is similar to `\InRange` in that it is a pseudo-value to be used for the second argument of `\Expect`. It makes `\Expect` succeed if its own mandatory argument is close to the first argument from `\Expect`, where closeness is defined as being within 0.05pt. This size can be varied by specifying a different one as optional argument to `\NearTo`. Here is a test:

```
(*etex)
\begin{qstest}{\NearTo success}
  {\NearTo}
  \dimen@=10pt
  \Expect*{\the\dimen@}
    \NearTo{10.05pt}
  \Expect*{\the\dimen@}
    \NearTo{9.95pt}
  \Expect*{\the\dimen@}
    \NearTo[2pt]{12pt}
  \Expect*{\the\dimen@}
    \NearTo[0.1pt]{9.9pt}
\end{qstest}
\begin{qstest}{\NearTo failure}
  {\NearTo, fails}
  \dimen@=10pt
  \Expect*{\the\dimen@}
    \NearTo{10.05002pt}
  \Expect*{\the\dimen@}
    \NearTo[1pt]{11.00001pt}
\end{qstest}
</etex>
```

1.6 Saved results

The purpose of saved results is to be able to check that the value has remained the same over multiple passes. Results are given a unique label name and are written to an auxiliary file where they can be read in for the sake of comparison. One can use the normal `aux` file for this purpose, but it might be preferable to use a separate dedicated file. That

way it is possible to input a versioned *copy* of this file and have a fixed point of reference rather than the last run.

While the `aux` file is read in automatically at the beginning of the document, this does not happen with explicitly named files. You have to read them in yourself, preferably using

```
\InputIfFileExists
{filename}{-}{-}
```

so that no error is thrown when the file does not yet exist.

`\SaveValueFile` gets one argument specifying which file name to use for saving results. If this is specified, a special file is opened. If `\SaveValueFile` is not called, the standard `aux` file is used instead, but then you can only save values after `\begin{document}`. `\jobname.qsout` seems like a useful file name to use here (the extension `out` is already in use by `pdfTeX`).

```
\begin{qstest}{\SavedValue}
           {\SavedValue}
\SaveValueFile{\jobname.qsout}
```

If this were a real test instead of just documentation, we probably would have written something like

```
\InputIfFileExists
{\jobname.qsin}{-}{-}
```

first in order to read in values from a previous run. The given file would have been a copy of a previous `qsout` file, possibly checked into version control in order to make sure behavior is consistent across runs. If it is an error to not have such a file (once you have established appropriate testing), you can just write

```
\input{\jobname.qsin}
```

instead, of course.

`\CloseValueFile` takes no argument and will close a value save file if one is open (using this has no effect if no file has been opened and values are saved on the `aux` file instead). We'll demonstrate use of it later. It is probably only necessary for testing `qstest` itself (namely, when you read in saved values in the same run), or when you do the processing/comparison with a previous version by executing commands via `TeX`'s `write18` mechanism.

`\SaveValue` gets the label name as first argument. If you are using the non- ϵ -`TeX` version, the label name gets sanitized using `\string` and so can't deal with non-character material except at its immediate beginning. The ϵ -`TeX` version has no such constraint.

The second argument is the same kind of argument as `\Expect` expects, namely something suitable for a token register assignment which is passed

through `\def` if not preceded by `*`, and by `\edef` if preceded by `*`. The value is written out to the save file where it can be read in afterwards.

Let us save a few values under different names now:

```
\SaveValue{\InternalSetValue}
           *{\meaning\InternalSetValue}
\SaveValue{\IncludeTests}
           *{\meaning\IncludeTests}
\SaveValue{whatever}
           *{3.1415}
\SaveValue{maxdimen}
           *{\the\maxdimen}
```

A call to `\InternalSetValue` is placed into the save file for each call of `\SaveValue`. The details are not really relevant: in case you run into problems while inputting the save file, it might be nice to know.

`\SavedValue` is used for retrieving a saved value. When used as the second argument to `\Expect`, it will default to the value of the first argument to `\Expect` unless it has been read in from a save file. This behavior is intended for making it easy to add tests and saved values and not get errors at first, until actually values from a previous test become available.

Consequently, the following tests will all turn out true before we read in a file, simply because all the saved values are not yet defined and default to the expectations:

```
\Expect{Whatever}
           \SavedValue{\InternalSetValue}
\Expect[\IncludeTests]{Whatever else}
           \SavedValue{\IncludeTests}
\Expect[whatever]{2.71828}
           \SavedValue{whatever}
\Expect[undefined]{1.618034}
           \SavedValue{undefined}
```

After closing and rereading the file, we'll need to be more careful with our expectations, but the last line still works since there still is no saved value for "undefined".

```
\CloseValueFile
\input{\jobname.qsout}
\Expect*{\meaning\InternalSetValue}
           \SavedValue{\InternalSetValue}
\Expect[\IncludeTests]
           *{\meaning\IncludeTests}%
           \SavedValue{\IncludeTests}
\Expect[whatever]{3.1415}
           \SavedValue{whatever}
\Expect[undefined]{1.618034}
           \SavedValue{undefined}
\end{qstest}
```


Now let's take the previous tests which succeeded again and let them fail now that the variables are defined:

```
\begin{qstest}{\SavedValue failure}
      {\SavedValue, fails}
  \Expect{Whatever}
      \SavedValue{\InternalSetValue}
  \Expect[\IncludeTests]{Whatever else}
      \SavedValue{\IncludeTests}
  \Expect{2.71828}\SavedValue{whatever}
\end{qstest}
```

1.7 Checking for call sequences

The environment `ExpectCallSequence` tells the test system that macros are going to be called in a certain order and with particular types of arguments.

It gets as an argument the expected call sequence. The call sequence contains entries that look like a macro definition: starting with the macro name followed with a macro argument list and a brace-enclosed substitution text that gets executed in place of the macro. The argument list given to this macro substitution will get as its first argument a macro with the original definition of the control sequence, so you can get at the original arguments for this particular macro call starting with `#2`. As a consequence, if you specify no arguments at all and an empty replacement text for the substitution, the original macro gets executed with the original arguments.

`\CalledName`: if you want to get back from the control sequence with the original meaning in `#1` to the original macro name, you can use `\CalledName` on it. This will expand to the original control sequence *name*, all in printable characters fit for output or typesetting in a typewriter font (or use in `\csname`), but without the leading backslash character. You can get to the control sequence itself by using

```
\csname \CalledName#1\endcsname
```

and to a printable version including backslash by using

```
\expandafter \string
\csname \CalledName#1\endcsname
```

Going into more detail, a substitution function is basically defined using

```
\protected \long \def
```

so it will not usually get expanded except when hit with `\expandafter` or actually being executed. Note that you can't use this on stuff that has to work at expansion time. This works mainly with macros that would also be suitable candidates for `\DeclareRobustCommand`.

It is also a bad idea to trace a conditional in this manner: while the substitution could be made to work when being executed, it will appear like an ordinary macro when being skipped, disturbing the conditional nesting.

Only macros occurring somewhere in the call sequence will get tracked, other macros are not affected. The environment can actually get nested, in which case the outer sequences will get tracked independently from the inner sequence.

Thus, `ExpectCallSequence` can be used in order to spoof, for example, both input consuming and output producing macros without knowing the exact relationship of both.

Apart from specifying macro calls, the call sequence specification can contain the following characters that also carry a special meaning:

' If this is set in the call sequence, it places the initial parsing state here. This will make it an error if non-matching entries occur at the start of the sequence, which otherwise is effectively enclosed with

```
.{ }*(sequence).{ }*
```

meaning that nonmatching entries before the first and after the last matching item of the sequence are ignored by default (this makes it closer to normal regexp matchers). Since the matching will then start at `'`, you can put macros before that position that you want to be flagged if they occur in the sequence, even when they are mentioned nowhere else (macros which would be an error if actually called). Also available as the more customary `^` character, but that tends to behave worse in L^AT_EX-aware editors.

' This indicates the last call sequence element to be matched. If any traced macros appear after this point, an error will get generated. Any immediately following call sequence entries will not get reached.

. A single dot indicates a wildcard: any of the tracked control sequences might occur here. You still have to follow this with macro arguments and a braced replacement text. Wildcards are considered as a fallback when nothing else matches.

(...) Parens may be used for grouping alternatives and/or combining items for the sake of repeating specifications, of which there are three:

? If a question mark follows either a macro call, wildcard call, parenthesized group, or call sequence end, the item before it is optional.

- + A plus sign following an item means that this item may be repeated one or more times.
- * An asterisk following an item means that this item may be repeated zero or more times.
- | A vertical bar separates alternatives. Alternatives extend as far as possible, to the next bar, to an enclosing paren group, or to the start and/or end of the whole call sequence specification if nothing else intervenes.

Note that in contrast to traditional regexp evaluation, no backtracking is employed: at each point in the call sequence, the next match is immediately chosen and a choice cannot (for obvious reasons) be reverted. It is the task of the user to specify a call sequence in a sufficiently non-ambiguous manner that will make the call sequence tracing not pick dead ends.

```

\begin{qstest}{ExpectCallSequence}
    {ExpectCallSequence}
    \def\{e\} \def\{f\}
    \def\{g\} \def\{h\}
    \begin{ExpectCallSequence}
        {\e#1{%
            \Expect\expandafter
            {\csname\CalledName#1\endcsname}
            {\e }%
            \Expect*{\meaning#1}
            {macro:->e}}+\f#1{}}
        \e \e \e \e \f
    \end{ExpectCallSequence}
\end{qstest}

```

1.8 Ending a standalone test file

One finishes a standalone test file by calling the \LaTeX control sequence `\quit`. This stops processing even when we did not actually get into a document. We don't actually do this here since there will be further tests in the full documentation file. However, we will now close the log file we opened for this demonstration.

```

\LogClose{lgout}
</test>

```

2 Conclusion

The package documentation illustrated how one can embed test cases into the source of a `dtx` package by using module guards `<test>` and `docstrip`. There are more possibilities of use, such as using `<trace>` guards and embedding `\Expect` macros and call sequence expectations right into code for regular use instead of doing separate tests. In that way, a debugging version of the package may be extracted using `docstrip`. Selecting a subset of trace commands or assertions to use can easily be accomplished with the `makematch` package.

The `qstest` package in combination with the `dtx` documentation format and `docstrip` allows to integrate documentation and unit testing. As long as one does not do actual testing, the `qstest` package is not required to be installed for either compiling documentation or using the style file. For that reason, one can safely use it without having to assume anything about the version (if any) of the `qstest` package available to some end user.

Grzegorz Murzynowski
 Sulejówek, Poland
 natror at o2 dot pl

Abstract

I wrote a package for ‘optical centering’ of verses and for right alignment of long and broken lines. By ‘optical centering’ I mean placement of the box containing a verse such that it seems to be balanced on the vertical axis of a page.

Another package of mine is gmcontinuo, working under both L^AT_EX and Plain T_EX. It allows one to typeset paragraphs *in continuo*, marked not with a new line and indent but continuously, marked with only the ¶ sign.

1 The gmverse package

1.1 Alignment of broken lines

Apentula niewdziosek te będą
 gruwaśne
 W kość turmiela weprzącznie, kostrą
 bajtę spoczy

Figure 1: A verse by S. Lem in a standard verse ...

Oproszędy zniemęci, wyświrle
 uwzroczy,
 A korśliwe porsacze dogremnie
 wyczańnie.

Figure 2: ... and in verse of gmverse: long and broken lines aligned right ...

Trzy samolóz wywiorstne gręczacz
 [tęci wzdyżmy,
 Apelajda sękliwa borowajkę
 |kuci...

Figure 3: ... and bracketed.

As you can see in figs. 2 and 3, gmverse provides for right alignment of long and broken lines. This seems to be typical in Polish typography and one gets that effect with just `\usepackage{gmverse}`. Optionally, the rest of a long line may be preceded with a left square bracket, which is used in Polish typesetting of poetry. That is specified with the `squarebr` option of gmverse.

What are the ‘long lines’, though? They are the lines exceeding `\hsize` or the length set with the `\versemaxline` declaration.

By the way, the source of the verse in fig. 3 is:

```
\begin{verse}
\l\Trzy\l samolóz\l wywiorstne\l (...)\l wzdyżmy,
\l\l\l Apelajda\l sękliwa\l borowajkę\l kuci\l\dots
\end{verse}
```

Do you see what’s missing? Yes. When I typeset poetry, I don’t want to think about `\l`, especially when I copy the verses from non-T_EX files. But if you insist, you may write `\l\`, it has the same effect and optional argument as in standard verses. A stanza you mark with a blank line.

Sometimes I typeset liturgical texts (psalms of Liturgia Horarum e.g.) that need to have stanzas (versetti) alternately indented. That is available in the `\psalmato` declaration’s scope, cf. fig. 4.

Składniki: cukier, miazga kakaowa, †
 migdały sześć procent, masło kakaowe *
 tłuszcz mleczny, gruszka, jabłko, ananas.
 Aromat naturalny gruszkowy, *
 emulgator lecytyna sojowa.
 Substancja zagęszczająca *
 alginian sodu.
 Regulator kwasowości kwas cytrynowy *
 E trzysta trzydzieści...

Figure 4: The `\psalmato` declaration’s effect.

(The cross and star indicate a continuation and division of a psalm line, respectively.)

1.2 Centerings

The gmverse package provides several kinds of centering of a verse. By centering of a verse I mean horizontal centering of some rectangle on a page. That rectangle corresponds to the body of a verse; *how* it corresponds depends on the kind of centering. The common feature of those rectangles is that their left side sticks to the left margin of the verse’s body. The verse itself is always aligned mostly left (ragged right and continuations of broken lines aligned right).

The optical centering may be either automatic or manual. The automatic comes in four flavours, all

² Anne-Sophie Mutter.

W_∞ :	ASM ² rządzi dioda świeci miodowo chrząszcz chrzęści w czcionkach.
W_3 :	ASM rządzi dioda świeci miodowo chrząszcz chrzęści w czcionkach.
W_2 :	ASM rządzi dioda świeci miodowo chrząszcz chrzęści w czcionkach.
W_1 :	ASM rządzi dioda świeci miodowo chrząszcz chrzęści w czcionkach.

Figure 5: Four versions of optical centering of a verse — E. Szarzyński, *Late night haiku No. 3*.

of them cases of a general formula. The idea is to compute some average of the lines’ lengths and set the rectangle’s width to that average.

The first kind of average one can think of is arithmetical mean. That is the case W_1 . The last kind of average is ‘only the longest line counts’, that’s W_∞ . Between them are a continuum of possibilities; consider a formula

$$W_\alpha = \frac{\sum_{k=1}^n l_k^\alpha}{\sum_{k=1}^n l_k^{\alpha-1}}$$

where l_k , $k = 1, \dots, n$ are the lengths of the lines³ and $\alpha \in [1, +\infty)$.

When $\alpha = 1$, all the lines have ‘equal right to decide’ about the mean. When α grows, the longer lines become ‘równiejsze’ (‘more equal’, a term from the PRL⁴ epoch to describe unjust privileges of the Party apparatchiks), which means they mean more to the mean.

We define W_∞ most naturally, as the limit:

$$W_\infty = \lim_{\alpha \rightarrow \infty} W_\alpha$$

and we notice easily that W_α corresponds to taking only the longest line into account.

³ Assume they all have nonzero length when $\alpha = 1$.

⁴ Polska Rzeczpospolita Ludowa (People’s Republic of Poland)

I personally like W_3 most and that’s the default when optical centering is on.

If you prefer to set the centering manually, you are given two ways to do that: the `\vocpussyhair` parameter (dimen) intended for slight modifications of the result of automatic computation, and the declaration(s)

```
\versecenterdue{\benchmark text}
```

or

```
\versecenterdue*{\benchmark dimen}.
```

2 The gmcontinuo package

One day I read on GUST-L a post posted by a non-newbie, who stated that *in continuo* typesetting was not possible in T_EX “because of the very nature of paragraphs in T_EX”. My immediate response to that was a package that works both in Plain and in L^AT_EX, which *does* do that: the text

¶ ... wyrosła budowla złożona i przedstawiająca i będąca marmurowo czarnymi (z czarnego marmuru) schodami prowadzącymi we Wszystkich kierunkach i zwrotach (rzecz jasna też poza tę czasoprzestrzeń). ¶ Z ust lampy białej jak gniazdo osy wypadła Salamandra i na pierwszych czterech stopniach budowli złożyła po jednym pomarańczowym jajku (ich powierzchnia była jak skórka pomarańczy). ¶ Przeliczyłem je — siedem z dołu w prawo, pięć z tyłu wglądź. ¶ Rzeczywiście, wszędzie wirują z wiatrem oka z pawich ogonów (niektóre mrugają porozumiewawczo)... ¶ *Edmund Szarzyński*.

is typeset from an ordinary T_EXt with paragraphs marked just with blank lines.

This small package is an example of a general rule that in T_EX you should never *ever* state “It’s impossible”, and you should not even ask “Can it be done?”, but: “*How* can it be done?”

3 The point

Why do I write in one article about such different things as right alignment of broken lines of a verse and *in continuo*? Because they use the same T_EX mechanism: unpacking of a once-typeset box, thanks to `\unhbox` and/or `\unvbox`, that lets you to retypeset a paragraph having found its dimensions.

The gmdoc bundle — a new tool for documenting (L^A)T_EX sources

Grzegorz Murzynowski

Sulejówek

natror at o2 dot pl

Abstract

There is a new package and a document class written by myself for documenting (L^A)T_EX packages and classes. ‘Documenting’ means that the comments are typeset as ordinary text and the code verbatim. All the control sequences are automatically indexed.

I think that the `gmdoc` package is superior to the `doc` package in two respects. First, the index entries, the table of contents and cross-references are made hyperlinks by default (with use of the `hyperref` package). Second, the `gmdoc` package allows you to typeset plain `.sty` and `.cls` files with the comments marked only with `%` (no special environments required).

The `gmdoc` bundle allows you to typeset the ‘traditional’ `.dtx` files, including L^AT_EX 2_ε `Source` and `doc.dtx`. The `gmdoc` bundle is available on CTAN.

gmdoc breaks free from macrocode

After I had written a couple of L^AT_EX packages and even a class, I realised it would be nice to document them and make them available for everybody by putting them on CTAN. So I asked my T_EX Guru, how can I document the code? I had already heard of the ideas of literate programming and self-documenting files. That idea is to write the code and the commentary on it simultaneously and mixed in one file, from which a respective program would extract the pure working code and another program would typeset a pretty narrated book or article about the code in question. Even before asking my T_EX Guru, I had always added much commentary to my code, T_EXnical or otherwise.

And my T_EX Guru told me a fascinating tale of the `doc` package, and the `.dtx` files that make possible literate programming of (L^A)T_EX sources. The main idea, of changing the catcode of `%` depending on the mode of reading a file, or, from another side, of allowing the same file to be an executable (loadable) package or document class or a comprehensive documentation of that package or class depending on the catcode of `%`, enlightened my mind. But the rest of the tale, although equally fascinating, suggested that I do something I wouldn’t like: mark up every piece of code with

```
%\begin{macrocode}  
...  
%\end{macrocode}
```

where the Percent and Four Spaces at the end are obligatory (see fig. 1). That would mean rewriting all of my `.sty` and `.cls` files.

Instead of such half-mechanical editorial work I chose to write my own documenting package such that just the percents would be sufficient as the markup, as in figure 2. Don’t you think that three lines of commentary instead of seven do make a difference and are more readable?

So, the task was set: not to mark up the code. The most natural¹ solution to that was the active line end which could check whether the next line begins with a comment sign or not.

The fundamental idea of `gmdoc` is to consider the input file as consisting of two threads: the commentary, marked with the comment signs, and the code, which is the rest of the file.

Therefore the first thing done by the main input command is setting the catcode of the declared comment sign (`%` by default) to ‘other’ (12) and the catcode of `~M` (the line end char) to ‘active’ (13) and define the newly-active line end to check whether the next line begins with the comment sign.

To be precise, that active line end memorizes the number of leading spaces of the next line and *then* checks whether the first non-space character is the comment sign. (Later, if we discover that it’s code, those spaces will be typeset as a respective indent.)

If the first non-space character of a line is not the comment sign, then the active line end opens a group for typesetting the code, within which the typewriter font is set, the catcodes of special characters are changed to 12 (‘other’) or 13 (‘active’) and

¹ I realize that what seems ‘most natural’ to me, may seem ‘Against Nature’ to some others ;-).

```

%uuuu\begin{macrocode}
\def\macrocode{\macro@code
%uuuu\end{macrocode}
%uuuuThenwetakecarethatallspaceshavethesamewidth,andthat
%uuuutheyarenotdiscarded.
%uuuu\begin{macrocode}
uuu\frenchspacing\@vobeyspaces
%uuuu\end{macrocode}
%uuuuBeforeclosing,weneedtocall|\xmacro@code|.uuItisthis

```

Figure 1: An excerpt from a .dtx file

```

\def\macrocode{\macro@code
uu%Thenwetakecarethatallspaceshavethesamewidth,andthat
uu%theyarenotdiscarded.
uuu\frenchspacing\@vobeyspaces%maybeaninlinecomment:
uuu%Beforeclosing,weneedtocall|\xmacro@code|.uuItisthis

```

Figure 2: An example of the desired markup

the characters redefined in the latter case. Then an iterating macro is launched that eats the code character by character and typesets it until it finds the comment sign.

In the last case, the macro checks whether it's a real beginning of a commentary and not just a concatenation of two lines of code, and if so (it's a commentary), it closes the verbatim group and lets the commentary be typeset.

In the comment or 'narration' layer, the comment char's catcode is set to 'ignored' (9), as with `doc`.

The solutions developed make `gmdoc` superior to the `doc` package in one and a half respects:

1. The `macrocode` environment is not compulsory anymore. It is available, however.
- 1.5. Inline comments are supported. That is, they are not typeset verbatim, but in a roman font, as the comments should be IMO.

By the way, don't you find the `gmdocish` version of a source (fig. 2) more readable (than in fig. 1)?

Usage

We haven't yet seen *how* the package should be used. The usage is very simple and analogous to the usage of `doc`: you write a usual \LaTeX document with some input commands specific to `gmdoc`, usually `\DocInput{<file.sty>}` or `\DocInclude{<file>}`; cf. fig. 3. That \LaTeX document file is called *the driver* (as in `doc`).

The text typeset in a roman font belongs to the narration layer, that is, it occurs after some %

sign. (As you might guess, the lines are numbered automatically.)

gmdoc meets hyperref

Since I've been into \TeX for only some three years, `.pdf` is a most natural output IMO and `pdf ϵ -TeX` is the most natural \TeX engine (though the marvellous `X ϵ TeX` may become so soon). So, an obligatory and almost subconscious behaviour is to use the `hyperref` package.

The sophisticated features of `doc`, such as automatic indexing of the control sequences and marking them in the margin seemed to me so useful and clever that I implemented them in `gmdoc`. And the features that I consider as 'naturally hyperlinking' are indeed made hyperlinks: the index entries, the cross-references, the footnotes, and the table of contents entries.

That's the other thing that makes `gmdoc` superior to `doc` IMO.

The TOC entries, the footnotes and the cross-references are made hyperlinks by default whenever you use the `hyperref` package. Therefore these features of `gmdoc` needed no work of mine (except

```
\RequirePackage{hyperref}.
```

The fourth thing, hyperlinking of the index entries, did need some care. By default, `hyperref` wants to make a hyperindex and that's very nice in most cases. But the case of documenting a (\LaTeX) \TeX source is different: The index entries may be of three kinds, two of which are specially formatted, and may be preceded with a source file identifier (here I follow

```

1 \documentclass[fleqn]{ltugproc}
2 \def\fileversion{\relax}
3 \hfuzz4pt
4 \PrelimDraftfalse
5 \tolerance990
6 \pretolerance1450
7 \input../lsetup.tex
8 \setcounter{page}{85}
9 \parskip0pt\plus.4pt
10 \usepackage{gmdoc}

```

This is a comment written as a separate paragraph. (The code is an excerpt from the source of this document.)

(...)

```
11 \begin{document}
```

(...)

The output of fig. 2:

```
12 \def\macrocode{\macro@code
```

Then we take care that all spaces have the same width, and that they are not discarded.

```
13 \frenchspacing\@vobeyspaces% maybe an inline comment: Before closing, we need to call
    \xmacrocode. It is this
```

(...)

```
14 \DocInput{gmdocEBT.tex}
```

```
15 And this is an example of a~very~long~code~line. See how is it {broken {%
    at {left {brace {with {a~\%sign as 'hyphen' and hang-indented.}}}}}
```

(...)

Figure 3: An example of use and output at once

the rules set by doc and ltxdoc, which I consider to be a (high) standard).

The need to use special encapsulation commands is obvious and that conflicts with the default |hyperpage encapsulation inserted by hyperref. So the appropriate encapsulations were written and now I dare say the high standard of a three-way² indexing of the CSs set by doc, along with the high standard of preceding the entries with the source file identifier when the source consists of several files set by ltxdoc, are wed to hyperref in gmdoc and the marriage is consummated.

Finishing touches

The preceding sections describe the two main ideas of gmdoc. The rest of the bundle I would call finishing touches. And they are many; I'll mention only few of them.

The gmdoc package provides hooks for the beginning and end of the input: `\AtBeginInput{initial`

² ;-).

`stuff to be added)}` and `\AtEndInput{finishing stuff to be added}`. Both use the ‘adding to a macro’ trick so multiple instances are allowed and accumulate. Both act globally.

But I also needed a hook that would add something only once, to the next input file. Therefore I wrote `\AtBeginInputOnce{the stuff}` hook that defines a macro of a unique name, thanks to

```
\csname...\the\some@count\endcsname
```

and the first thing the meaning of that macro consists of is `\let\this@macro\relax`, if you get what I mean, and then *the stuff*, of course.

The `\IndexInput` command analogous to doc’s homonym is crafted very simply: it consists mostly of the basic `\DocInput`, only the comment char, the code delimiter that is, is declared *char1*. Since *char1* is declared ‘invalid’ in L^AT_EX, we don’t expect one to be in a source file. Therefore the entire contents of a source file is considered to be the code, and typeset verbatim with its CSs automatically indexed.

In this command there is clearly visible a detail not that clear in ‘ordinary’ `\DocInput`: we have to put our code delimiter at the end of the input to be sure there are none in the file itself. But we do the same in `\DocInput` since we don’t want to require that a source file be ended with `%`.

There are a couple of commands for nicely typesetting CSs and their arguments. They are inspired by `doc`’s analogs, but defined in my own way. For instance, my `\cs{cs}` typesets `\cs` as expected, but also allows an optional argument, `\` by default, that is typeset before its mandatory argument. Thus, you may get `!macro` by writing `\cs[!]{macro}`. Why not just `\verb` or a ‘short verb’? Remember, that neither `\verb` nor ‘short verb’ can be used in an argument of a macro, nor can they be written to a file properly. And `\cs` is robust.

To get *(a meta-symbol)* I took the `\<...>` macro from *The T_EXbook* (and mixed it with `(ltx)doc`’s `\meta`). I mean, to get *(a meta-symbol)* you write `\<a^meta-symbol>`.

Moreover, for typesetting *{(arguments like this)}*, I defined the `\arg` command my way such that

code	typesets
<code>\arg_␣x=\pi\$</code>	<code>arg x = π</code>
<code>\arg{arg1}</code>	<code>{(arg1)}</code>
<code>\arg[optional]</code>	<code>[(optional)]</code>
<code>\arg(pictorial)</code>	<code>((pictorial))</code>

I also repeat a handful of logos provided in `doc` and add my ‘drei Groschen’:

<code>\AmSTeX</code>	$\mathcal{A}\mathcal{M}\mathcal{S}$ -T _E X
<code>\BibTeX</code>	BIBT _E X
<code>\SliTeX</code>	SLIT _E X
<code>\PlainTeX</code>	PLAIN T _E X
<code>\Web</code>	WEB
<code>\TeXbook</code>	<i>The T_EXbook</i>
<code>\eTeX</code>	ε -T _E X
<code>\pdfeTeX</code>	pdf ε -T _E X
<code>\pdfTeX</code>	pdfT _E X
<code>\XeTeX</code>	X _Ǝ T _E X
<code>\LaTeXpar</code>	(L ^A)T _E X
<code>\ds</code>	DocStrip

The first E in X_ƎT_EX is reversed if the `graphics` package is loaded. The (L^A)T_EX logo is defined in `gmutils` and therefore available independent of `gmdoc`.

I allow for a given source file to be typeset both standalone and as part of a multi-file document (The Great Anthology of My Œuvres for instance ;-) and therefore I provide ‘relative’ sectioning commands: `\division` and `\subdivision` are `\let` to `\section` and `\subsection` respectively but may be assigned another way in *The Anthology*.

Since my goal is for `gmdoc` to support both the standard classes and my favourite `mwcls`, in `gmutils` I cheat a bit about the sectioning commands to deal with their optional arguments in both the standard classes and `mwcls`.

Since I often use the Quasi-Fonts (now renamed and updated in T_EX Gyre) in the QX encoding,³ which doesn’t have the `␣` sign and that sign is needed when I wish the spaces in a verbatim environment⁴ to be ‘visible’, I added a hook to be executed (expanded) in every verbatim, after setting the catcodes and font. The contents of this hook, if you declare `\VerbT1`, is

```
\fontencoding{T1}\selectfont
```

so a visible space is typeset despite the general font encoding.

As in `doc`, you may declare some character(s) as ‘short verbatim’ and then write e.g. `|\verb␣` instead of `\verb*+\verb␣`. In fact, it’s not `gmdoc.sty` which makes it possible but `gmverb.sty`, so you may use that feature independent of `gmdoc`.

I prefer shorter markup to longer so to display single lines of code,

```
such␣\as␣\THIS␣one,
```

I redefined `\[` to make it properly typeset a short verbatim and spaces. So, you may type

```
\[|such␣\as␣\THIS␣one,|\]
```

to get the above.

I also wrote a document class to typeset the code in a pretty way, `gmdocc.cls`. This class is strongly inspired by the `ltxdoc` class but, again, it’s not a mere transcription.

In this article there’s not room to discuss all the features of this class so let’s look at a sample of output (see next page). Please notice the Latin Modern Typewriter Condensed on the margin (hope you like it as I do).

I could write many more words about what I consider the finishing touches. There are many options, declarations and commands to make documenting of sources as much comfortable as a princess could expect.

Approximately 87.31% of those touches were written to make the `gmdoc` bundle compatible with `doc` and `ltxdoc`, that is, to make `gmdoc` typeset the L^AT_EX canon of scriptures. And that leads us to the last part of this article.

³ Why do I use QX? I don’t remember, to be honest.

⁴ I mean all the verbatim-like commands: not only `verbatim`, but also the ‘shortverbs’ and the groups for the T_EX code in `gmdoc`.

The `\code@delim` should be `_12` so a space is not allowed as a code delimiter. I don't think it *really* to be a limitation.

And let's assume you do as we all do:

```
46 \CodeDelim%
```

We'll play with `\everypar`, a bit, and if you use such things as the `{itemize}` environment, an error would occur if we didn't store the previous value of `\everypar` and didn't restore it at return to the narration. So let's assign a `\toks` list to store the original `\everypar`.

```
47 \newtoks\gmd@preverypar
```

```
48 \newcommand*\settetcodehangi{%
```

```
49   \hangindent=\verbatimhangindent_\hangafter=\@one}% we'll use it in the inline
      comment case. \verbatimhangindent is provided by the gmverb package
      and = 3em by default.
```

```
50 \@ifdefinable\@settetcodehangi{\let\@settetcodehangi=%
      \settetcodehangi}
```

We'll play a bit with `\leftskip`, so let the user have a parameter instead. For normal text (i.e. the comment):

```
\TextIndent 51 \newlength\TextIndent
```

I assume it's originally equal to `\leftskip`, i.e. `\z@`. And for the T_EX code:

```
52 \newlength\CodeIndent
```

```
\CodeIndent 53 \CodeIndent=1,5em\relax
```

And the vertical space to be inserted where there are blank lines in the source code:

```
54 \@ifundefined{stanzaskip}{\newlength\stanzaskip}{}
```

I use `\stanzaskip` in `gmverse` package and derivatives for typesetting poetry. A computer program code *is* poetry.

```
\stanzaskip 55 \stanzaskip=\medskipamount
```

```
56 \advance\stanzaskip_\by-.25\medskipamount% to preserve the stretch- and shrink-
      ability.
```

A vertical space between the commentary and the code seems to enhance readability so declare

```
57 \newskip\CodeTopsep
```

```
58 \newskip\MacroTopsep
```

And let's set them. For aesthetic minimality⁷ let's unify them and the other most important vertical spaces used in `gmdoc`. I think a macro that gathers all these assignments may be handy.

```
\UniformSkips 59 \def\UniformSkips{%
```

```
\CodeTopsep 60   \CodeTopsep=\stanzaskip
```

```
\MacroTopsep 61   \MacroTopsep=\stanzaskip
```

```
62   \abovedisplayskip=\stanzaskip
```

`\abovedisplayshortskip` remains untouched as it is 0.0 pt plus 3.0 pt by default.

```
63   \belowdisplayskip=\stanzaskip
```

⁷ The terms 'minimal' and 'minimalist' used in `gmdoc` are among others inspired by the *South Park* cartoon's episode *Mr. Hankey The Christmas (...)* in which 'Philip Glass, a Minimalist New York composer' appears in a 'non-denominational non-offensive Christmas play' ;-). (Philip Glass composed the music to the *Qatsi* trilogy among others)

Figure 4: A sample of `gmdoc` output

Testing or Missa papae Marcelli

In the 16th century there was a controversy in the Roman Catholic Church about polyphony. There is a legend that Pope Marcellus II considered banning it since many composers were making the texture of their works so complex that the words were not recognizable. Then Giovanni Pierluigi da Palestrina wrote a beautiful and ingenious polyphonic Missa whose texture is extremely dense but the words are very clearly recognizable. That missa, dedicated to the pope, convinced him to allow polyphony in the church music.

Why do I write about this? Because I hope the `gmdoc` bundle at least generates a controversy whether to use the `doc` package and the `ltxdoc` class or *itself*. To be honest, my hope is the `gmdoc` bundle could replace `doc` and `ltxdoc`. In a sense, `gmdoc` is compatible with them: it typesets ‘traditional’ `.dtx` files including The L^AT_EX 2_ε Source.

One has just to use `\OldDocInput` instead of `\DocInput` or declare `\olddocIncludes` before `\DocInclude` of a `docish` file.

The (working!) driver files for The Source and some other canonical files are my Missa papae Marcelli.

First, an homage to `doc` and `ltxdoc`, from which I took most of the ideas (although, as a rule, I didn’t copy the macros but rather made mine do what they do): `doc_gmdoc.tex`.

My esteem for those packages and classes is so deep that I didn’t report either of the two typos noticed during my typesetting nor did I change the original text, but wrote some ‘diving hooks’ to fix them.

Then, for their close relative, `docstrip.dtx`: `docstrip_gmdoc.tex`.

And, last and most thrilling, The L^AT_EX 2_ε Source: `source2e_gmdoc.tex`.

Those drivers are available on CTAN as a part of the `gmdoc` bundle.

I hope this humble bundle will be useful for someone else and not only for me.

Brave new version 0.99g

While preparing this article for *TUGboat*, I revised the `gmdoc` bundle and made it work with X_YL^AT_EX and automatically detect a couple of definitions.

‘Works with X_YL^AT_EX’ means that you can specify the `sysfonts` option of the `gmdoc` document class;

the basic three X_YL^AT_EX-related packages (`fontspec`, `xunicode` and `xltxtra`) will be loaded, and then you can specify the system fonts with the `fontspec` package declarations.

‘Automatically detects a couple of definitions’ means that if you use `gmdoc` with its default settings, any occurrence (in the code layer) of the defining commands listed below causes marking of their argument (the thing being defined) as defined at that point: the control sequence, environment, counter or option being defined appears in a margin note and is indexed as a ‘definition’ entry.

The detected commands are:

- the (L^A)T_EX standard definitions: `\def`, `\newcount`, `\newdimen`, `\newskip`, `\newif`, `\newtoks`, `\newbox`, `\newread`, `\newwrite`, `\newlength`, `\newcommand(*)`, `\renewcommand(*)`, `\providecommand(*)`, `\DeclareRobustCommand(*)`, `\DeclareTextCommand(*)`, `\DeclareTextCommandDefault(*)`, `\newenvironment(*)`, `\renewenvironment(*)`, `\DeclareOption(*)`, `\newcounter`;
- the definitions of the `xkeyval` package: `\define@key`, `\define@boolkey`, `\define@choicekey`, `\DeclareOptionX`;
- and the option definitions of the `kvoptions` package by Heiko Oberdiek: `\DeclareStringOption`, `\DeclareBoolOption`, `\DeclareComplementaryOption`, `\DeclareVoidOption`.

Moreover, if you have your own defining commands, they can now be detected with `\DeclareDefining⟨command⟩`. On the other hand, you can turn off the detection with `\HideDefining⟨command⟩` for the `⟨command⟩` only or `\HideAllDefining` for all the definitions.

There are further commands that allow resuming detection after ‘hiding’ it and particular declarations for `\def` since it does not always define an important macro.

And you still have the `\Define` declaration and the `macro(*)` environment if the automatic detection doesn’t fit your needs.

Concluding, the `gmdoc` bundle now makes possible typesetting of (L^A)T_EX sources with almost no markup and with the advantages of `hyperref` and X_YL^AT_EX.

T_EX beauties and oddities

A permanent call for T_EX pearls

<http://www.gust.org.pl/pearls>

What is wanted:

- ▷ short T_EX, METAFONT or MetaPost macro/macros (half an A4 page or half a screen at most),
- ▷ the code should be generic; potentially understandable by plain-oriented users,
- ▷ results need not be useful or serious, but language-specific, tricky, preferably non-obvious,
- ▷ obscure oddities, weird T_EX behaviour, dirty and risky tricks and traps are also welcome,
- ▷ the code should be explainable in a couple of minutes.

The already collected pearls can be found at <http://www.gust.org.pl/pearls>. All pearl-divers and pearl-growers are kindly asked to send pearl-candidates to pearls@gust.org.pl, where Paweł Jackowski, our pearl-collector, is waiting impatiently. The pearl marketplace is active during the entire year, not just before the annual BachoT_EX Conference.

Note: The person submitting pearl proposals and/or participating in the BachoT_EX pearls session need not be the inventor. Well known hints are also welcome, unless already presented at one of our sessions.

Since some seasoned T_EX programmers were indignant at calling ugly T_EX constructs “Pearls of T_EX programming”, we decided not to irritate them any longer. We hope they will accept “T_EX beauties and oddities” as the session title.

If you yourself have something that fits the bill, please consider. If you know somebody’s work that does, please let us know, we will contact the person. We await your contributions even if you are unable to attend the conference. In such a case you are free either to elect one of the participants to present your work or “leave the proof to the gentle reader” (cf., e.g., <http://www.aurora.edu/mathematics/bhaskara.htm>).

A T_EX quine (*Péter Szabó*)

The code producing itself:

```
\def\T{
\tt \hsize 32.5em\parindent 0pt\def \S {\def \S ##1>{}}\S \string
\def \string \T \string {\par \expandafter \S \meaning \T \string
}\par \expandafter \S \meaning \T \footline {} \end }
\tt \hsize 32.5em\parindent 0pt\def \S {\def \S ##1>{}}\S \string
\def \string \T \string {\par \expandafter \S \meaning \T \string
}\par \expandafter \S \meaning \T \footline {} \end
```

Multi-signed numbers (*Hans Hagen*)

T_EX handles multiple signs properly:

```
\newdimen\scratchdimen
\scratchdimen 1pt \the\scratchdimen,
\scratchdimen -1pt \the\scratchdimen,
\scratchdimen --1pt \the\scratchdimen,
\scratchdimen ---1pt \the\scratchdimen,
\scratchdimen -+--+---+-----+1pt \the\scratchdimen,
```

So, there is never a need to use an intermediate variable to negate a value. All digits, +/− signs and units can be faked in macros:

```
\def\neg{-} \def\p{p}
\scratchdimen \neg\space\neg\space\space00001\empty\p\empty\empty tttt
```

One may also notice that while whitespace characters are allowed between multiple signs (but not between digits!), leading zeros are ignored, and the unit is properly interpreted regardless of the very next character.

Double-hat trap (*Jerzy Ludwichowski*)

Is there a difference between those two cases?

```
\number'\^^A
\number'^^A
```

And how about this?

```
\number'\^^@
\number'^^@
```

In the case of ^^A (character code 1), both lines yield the number 1, the backslash character's presence before the double-hat doesn't influence the result. In the second case, the first line yields 0, while the second results in 32. The reason is that the character of the code 0 (^^@) has the associated category code 'ignored' (9). Any character of the category 9 will simply be omitted, except when there is a backslash immediately before it. If there is no backslash, the very next character is considered, which is a space (code 32), and ^^@ simply disappears. This does not happen with characters of category code different from 9.

\vbox height vs. \vtop depth (*Pawel Jackowski*)

\vbox usually inherits its depth from the last box or rule of the vertical list it contains. Conversely, \vtop has usually the height of the first box or rule of the vertical list it contains. However, using whatsits as the first/last item of the box may lead to surprises.

```
\def\what{\special{}}
\setbox0=\vbox{Aqq \what} \the\ht0, \the\dp0 % 6.83331pt, 1.94444pt
\setbox0=\vtop{\what Aqq} \the\ht0, \the\dp0 % 0.0pt, 8.77776pt
```

\vbox still obeys the rule, despite the whatsit after the very last box on the list. But \vtop always has zero height if its first item is a whatsit.

(Ir)relevant missing character message (*Paweł Jackowski*)

Try out the code

```
\hsize=7.3in \vsize=9.8in \leftskip=30mm \rightskip=30mm \parindent=1em
\font\LOGO=logo10
\def\MP{\LOGO METAPOST}
\def\MF{\LOGO METAFONT}
```

short \TeX, \MF\ or \MP\ macro/macros (half A4 page or half a-screen at most)[...]

The output is typeset without breaking any word at the end of a line. Try then to explain why the log file contains the line:

```
Missing character: There is no - (45) in font logo10!
```

While breaking paragraphs into lines T_EX checks all feasible breakpoints and chooses the one of the smallest sum of costs (see *The T_EXbook*, chapter 14). The message in the log file informs that some of the ways T_EX considered of typesetting the paragraph had a discretionary break after a META.

Skip assignments (*Paweł Jackowski*)

Consider the code:

```
\newskip\A
\newskip\B
\A = 3pt plus 1pt minus 1pt
\B = 1\A
```

Is now the skip \B equal to \A?

No, it's not:

```
\the\A % 3pt plus 1pt minus 1pt
\the\B % 3pt
```

In an assignment of the form

```
skip = <number> skip
```

T_EX eliminates the stretch and shrink of the glue. To avoid this effect one should not use a number/factor ('1' in this case) on the right hand side of the equation. When necessary, one should use the \advance, \divide, \multiply primitives instead, since all they preserve the glue-specific parts.

Current font global assignment (*Bogusław Jackowski*)

Font setup is normally bounded by groups. The code

```
\font\A=ec-lmr10 \A \message{\the\font}
{\font\B=ec-lmtt10 \B \message{\the\font}}
\message{\the\font}
```

gives \A \B \A, as one would expect. Why then does

```
\font\A=ec-lmr10 \A \message{\the\font}
{\font\B=ec-lmr10 \B \message{\the\font}}
\message{\the\font}
```

yield \A \B \B?

When the font used inside a group is the same as the current font in the outer grouping level, the local font assignment becomes global. In fact, font \A is internally mapped to \B. Even if we call \A explicitly, T_EX reports \B as the current font.

```
\A \message{\the\font}
```

Things are intentionally different in LuaT_EX ...

How to make a box disappear at a line break (*Marcin Woliński*)

Let us consider the problem of marking spaces in a paragraph with some symbol, as in the following:

Ten · typowy · testowy · akapit · tekstu · daje · przy · okazji · rodzaj
filigranowego · wysypu · hodowli · pieczarek · w · zielonym · kaszta-
nie · regloryfikacji · stanowisk · ministerialnych · i · podszypanych
minimalistom · jako · fetysz · zaduchu · studziennych · barykad.

The hard part is to make the symbol disappear when such a “space” occurs at a line break. We cannot use `\discretionary` for that purpose since we need the “space” to be stretchable to make justification possible. Moreover we want to be able to associate some penalty (e.g., 0) with our breakpoints other than `\(ex)hyphenpenalty`.

As it turns out any box can be made discardable by putting it into `\cleaders` to the exact width of the box in question. According to the rules T_EX will put exactly one copy of the box in the text. So the construct will look exactly like the box itself but will behave like a glob of glue. In particular it will disappear at a line break.

Here are the macros used in the preceding passage:

```
\obeyspaces
\def {%
\setbox0\hbox{${\cdot}$}%
\dimen0=\wd0\relax
\hskip1ptplus2pt%
\cleaders\box0\hskip\dimen0%
\hskip1ptplus2pt%
}
\rm\hsize9.5cm\parindent0pt
Ten typowy testowy akapit tekstu daje przy okazji rodzaj filigranowego %
wysypu hodowli pieczarek w zielonym kasztanie regloryfikacji %
stanowisk ministerialnych i podszypanych minimalistom jako fetysz %
zaduchu studziennych barykad.%
```

Stretchability is achieved with separate globs of glue so as not to allow T_EX to insert more than one copy of the box in case of an overstretched space.

Note that this trick can be used in vertical mode as well (e.g., to separate paragraphs with some graphical element except the case when a paragraph boundary occurs at a page break). A discardable box can have arbitrary complexity, it can include colour, EPS graphics, and so on.

Variable-width visible space (*Bogusław Jackowski*)

Marked spaces in a paragraph may not only disappear at a line break (as presented in the previous beauty by Marcin Woliński), but may also adjust their width, shrink and stretch, as normal interword space does.

```
\def\vispace{%
\ifdim\spaceskip=0pt
\skip0=\fontdimen2\the\font
plus \fontdimen3\the\font
minus \fontdimen4\the\font
\else \skip0=\spaceskip \fi
\advance\skip0-.4pt
\cleaders\vrule width.2pt height.2ex depth.2pt\hskip.2pt
\cleaders\hrule height0pt depth.2pt\hskip\skip0
\cleaders\vrule width.2pt height.2ex depth.2pt\hskip.2pt
}
```

```
\obeyspaces\let \=\vispace\def~{\nobreak\vispace}\let\ \=\vispace%
% \def\~M{\ } % plain does
```

Ten typowy testowy akapit tekstu daje przy okazji rodzaj filigranowego wysypu hodowli pieczarek w zielonym kasztanie regloryfikacji stanowisk ministerialnych i podsypanych minimalistom jako fetysz zaduchu studziennych barykad aglomeracji fosforescencji luminazy atraktywno-bajerywnej z dodatkiem glukozy i mineralnych bakterii finansowych oraz gromadzenia idei atrakcyjnych pomp prasowych z okazji rozpoczynania vegetacji takich istot jak wiolonczele, napoje bazaltowe i gramatyka z okresu mezozoicznego z jej typowym sposobem oznajmiania zachwytu nad bytem poprzez wycie i popiskiwanie o charakterystycznej modulacji toniczno-barycznej z wysokami w kierunku reglamentacji zawartej immanentnie w bagnie.

Ten typowy testowy akapit tekstu daje przy okazji rodzaj filigranowego wysypu hodowli pieczarek w zielonym kasztanie regloryfikacji stanowisk ministerialnych i podsypanych minimalistom jako fetysz zaduchu studziennych barykad aglomeracji fosforescencji luminazy atraktywno-bajerywnej z dodatkiem glukozy i mineralnych bakterii finansowych oraz gromadzenia idei atrakcyjnych pomp prasowych z okazji rozpoczynania vegetacji takich istot jak wiolonczele, napoje bazaltowe i gramatyka z okresu mezozoicznego z jej typowym sposobem oznajmiania zachwytu nad bytem poprzez wycie i popiskiwanie o charakterystycznej modulacji toniczno-barycznej z wysokami w kierunku reglamentacji zawartej immanentnie w bagnie.

Do you need some stretch? (*Marcin Woliński*)

T_EX's `\leaders` primitive can be used to fill arbitrary space with a stretchable line (cf. `\hrulefill`). It is also possible to have an expandable triple line:



```
\def\triplefil{%
  \leaders\hrule height4pt depth-3.2pt\hfil \hskip0pt plus-1fil
  \leaders\vrule height1.6pt depth0pt\hfil \hskip0pt plus-1fil
  \leaders\vrule height-.6pt depth1pt\hfil }
\def\triplefilledline#1{\hbox to\hsize{%
  \vrule height4ptdepth3ptwidth.8pt \triplefil \vrule
  height10ptdepth1ptwidth.4pt \enspace\strut#1\enspace \vrule
  height10ptdepth1ptwidth.4pt \triplefil \vrule
  height4ptdepth3ptwidth.8pt } }
\triplefilledline{The St.\ Anford Orchestra}
\triplefilledline{Variations on a Theme by Tchaikovsky}
```

To understand what happens here one needs to count stretchability of leaders and glue in `\triplefil`. It is: 1fil (from `\hfil`) + -1fil (from `\hskip`) + 1fil + -1fil + 1fil, which sums up to 1fil. So when T_EX needs to set `\triplefil` to, say, 37pt it stretches each fil of glue to that length. The first leaders become 37pt wide, then comes `\hskip` to -37pt (-1fil), and so T_EX overprints the second `\leaders` on the first, and the same repeats with the next glue and leaders.

This trick opens space for countless variations:



MetaPost tables indexed with strings (*Bogusław Jackowski*)

Converting MetaPost strings to suffixes one can implement tables indexed with strings.

```
% Definitions:
def strtosfx(expr s) =
for i:=1 upto length(s): [ASCII(substring(i-1,i) of s)] endfor
enddef;
vardef sfxtostr_ []@# =
if (str @=""): "" else: char(@) if str @#<>"": & (sfxtostr_ @#) fi fi
enddef;
def sfxtostr(suffix s) = begingroup sfxtostr_ s endgroup enddef;

% A few tests:
show sfxtostr(strtosfx("ABCABCABCABCABCABCABCABCABCABCABC!"));

save X; X strtosfx("ABC") =0; showvariable X;
save X;
for s:="ala", "ma", "kotakotakota", "kota": X strtosfx(s) = 0; endfor
for s:="ala", "ima", "kota": if known X strtosfx(s): show s; fi endfor
end.
```

If only there were a way to iterate over all known indexes ...

Multiple expansions triggered with a single `\expandafter` (*Marcin Woliński*)

This pearl (coded on October 18, 1996) is the most useless one I could think of. Nonetheless it is an example of a really curious expansion of macros.

Let us imagine that we have a list of non-space tokens and we want to assign this list to a token register without expanding the tokens and in reversed order. Here is a simple macro that reverses a list in an expand-only way:

```
\def\afterfi#1#2\fi{\fi#1}

\def\reverse#1{\reverseX{ }#1\stopreverse}
\def\stopreverse{\noexpand\stopreverse}

\def\reverseX#1#2{\ifx\stopreverse#2%
  \afterfi{#1}%
\else
  \afterfi{\reverseX{#2#1}}%
\fi}
```

Now we can write

```
\reverse{abcdefg}
```

and T_EX will respond with writing `gfedcba` on the terminal.

To put the result of reversing the list `abc\foo def\bar ghi` in a token register we do the following:

```
\toks0=\expandafter{\if0\reverse{abc\foo def\bar ghi0}}\fi
\showthe\toks0
```

With the use of `\expandafter` we introduce a single expansion to the region where expansion is suppressed. The token being expanded is the `\if`. To expand an `\if` T_EX needs to find the next two non-expandable tokens to compare them. The first token is `0`, but then T_EX sees the macro `\reverse`. So the macro gets expanded. An interesting feature of `\reverse` is that no non-expandable tokens are emitted until the list is fully reversed. So only then does T_EX stop expansion. The first non-expandable token T_EX will see is the second `0`, which we have devilishly inserted at the end of the list. At this point the condition turns out to be true and the next tokens get assigned as contents to the token register.

Hacking verbatim (*Grzegorz Murzynowski*)

How do you get *italics* inside a verbatim? By a ‘verbatim’ I mean a L^AT_EX environment that changes the catcodes of special chars and thus allows typesetting them verbatim (the tricks below apply to T_EX in general, though). L^AT_EX’s `\begin{verbatim}` expands mostly to `\begingroup\csname verbatim\endcsname` and `\verbatim` acts mostly like DEK’s `\ttverbatim`, `\end{verbatim}` is needed to delimit `\verbatim`’s argument.

Let’s recall that the chars of codes 1–32 (except the end of line, etc.) are catcoded as ‘invalid’ in L^AT_EX. Therefore I dare to assume they are neither used nor present in decent (L^A)T_EX files. The verbatim environments do not recatcode them, so I can use them for my wicked purpose:

```
\catcode'\^^E\active
\def^^E{\bgroup\it}
\let^^F\egroup
\begin{verbatim*}
How do you get <char5>italics<char6> inside a-verbatim?
\end{verbatim*}
```

Gives

```
How do you get italics inside a-verbatim?
```

Note that we should use explicit `<char5>` and `<char6>` since verbatims recatcode `^` to category ‘other’ so `^^E` would produce just `^^E`.

Now, how to input selected lines of a file verbatim?

```
\long\def\firstofone#1{#1}
\catcode'\@=11
\newread\my@file
\openin\my@file=bachotex2007-grzegorz-murzynowski-pearl1.src
\def\my@reading#1 #2{%
  \loop\ifnum\count\z@<#1%
    \advance\count\z@\@ne\read\my@file to\@tempa
    \ifx.#2\@tempa\endgraf\fi\repeat}%
\firstofone{%
  \begin{verbatim}%
  \count\z@\z@
  \my@reading1 -%
  \my@reading2 .%
  \my@reading22 -%
  \my@reading26 .%
}\end{verbatim}
```

The given code results in the following:

```
\def^^E{\bgroup\it}
\let^^F\egroup
\begin{verbatim*}
How do you get italics inside a-verbatim?
```

What is the most fundamental trick? The `\firstofone` macro (I learnt it from my T_EX Guru who did not invent it either). Apparently it doesn’t do anything: it has one parameter and expands exactly to it. But there is one very important thing it does: it ‘freezes’ the catcodes in the argument. Therefore all the commands and their arguments cannot be recatcoded by `\verbatim` and they are expanded and executed.

Custom overfull text (*Paweł Jackowski*)

How to replace a black overfull rule at the end of too long lines of a paragraph?

Well, there is no direct way to do so, but one should never underestimate T_EX's bells and whistles. First of all, we can test if the last (h)box was overfull by checking the value of `\badness`; if it is larger than 10000 it definitely means that the box was overfull (`\badness` never exceeds 10000 for underfull boxes). Assuming that `\box0` is the box we want to test, we can say

```
\def\oops{\hbox to\wd0{\setbox0=\hbox to\wd0{\unhbox0}%
\unhbox0 \ifnum\badness>10000 \rlap{\sevenrm\quad Oops!}\fi}}
```

And how to get the box that is the line of a paragraph? By setting the `\interlinepenalty` parameter to a large negative value we can force a page break between every two lines of a paragraph. In the `\output` routine, we can recognize those special penalties via the `\outputpenalty` parameter. The `\output` routine is not necessarily required to `\shipout` the page—it may simply return all its content back to the ‘recent contributions’.

```
\interlinepenalty=-50000 % force the break between each two lines
\maxdeadcycles=50        % allow upto 50 \outputs with no \shipout
\newtoks\orioutput \orioutput=\output % wrap the original \output routine
\output
{\ifnum\outputpenalty>-20000 \the\orioutput
\else \ifnum\outputpenalty<-\maxdimen \the\orioutput
\else
\unvbox255           % flush the entire list back
\setbox0=\lastbox % strip the very last box
\nointerlineskip % avoid doubled interline glue
\oops               % make the test and return the box back.
\advance\outputpenalty by50000
\penalty\outputpenalty % weak lie that nothing happened...
\fi\fi}
\hfuzz=\maxdimen % no overfullrule, no messages
\hsize=1.5in     % provoke overfulls
...
```

This completely useless example shows a not-so-useless trick, which `Oops!` might be used for quite advanced applications, such as line-numbering, `Oops!` some kind of paragraph decoration, page optimization and probably many others. Things become `Oops!` much more complicated if math displays, `\marks`, `\inserts` or `\va-` `Oops!` `djusts` come into play, but they don't spoil all of the game.

This completely useless example shows a not-so-useless trick, which might be used for quite advanced applications, such as line-numbering, some kind of paragraph decoration, page optimization and probably many others. Things become much more complicated if math displays, `\marks`, `\inserts` or `\va-` `djusts` come into play, but they don't spoil all of the game.

Crossword

Janusz Marian Nowacki

ul. Śniadeckich 82 m. 46

86-300 Grudziądz

Poland

janusz (at) jmn dot pl; www.jmn.pl

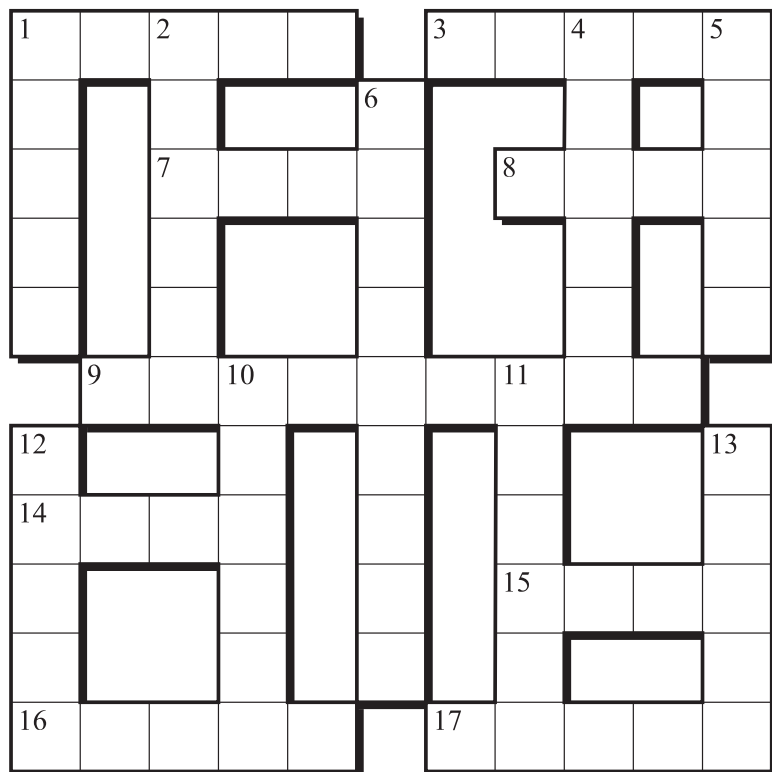
Across:

- 1) for saxophones and violins
- 3) horizontal law 7) before a code, apostrophe or double quote
- 8) a synonym to omit except in T_EX 9) an overused page layout
- 14) slashed of fractions or in PDF
- 15) a variant of the name Elisabeth
- 16) for fiddling with maths formulas
- 17) a pompous, stiff walk

Down:

- 1) don't on me 2) tells French from American 4) the last will not be the first but will be removed 5) vessels that make the greatest sound; also in English translation of "Riempi il bicchiere quando è vuoto, vuota il bicchiere quando è pieno, non lo lasciar mai vuoto, non lo lasciar mai pieno"
- 6) black glue 10) in abundance in one of the Bible books
- 11) anti-bump 12) integrated power supply frequency domain impedance meter 13) `\par\penalty-10000`

B.J., J.L., J.M.N., A.O.



Abstracts

LuaTeX: Messing around with tokens

Hans Hagen

Most TeX users only deal with (keyed in) characters and (produced) output. Some will play with boxes, skips and kerns, maybe even leaders (repeated sequences of the former). Others will be grateful that macro package writers take care of such things.

Macro writers on the other hand deal with properties of characters, like catcodes and a truckload of other codes, with lists made out of boxes, skips, kerns and penalties but even they cannot look much deeper into TeX's internals. Their deeper understanding comes from reading *The TeXbook* or even looking at the source code.

When someone enters the magic world of TeX and starts asking around a bit, he or she will at some point get confronted with the concept of tokens. A token is what ends up in TeX after characters have entered its machinery. Sometimes it even seems that one is only considered a qualified macro writer if one can talk the right token-speak. So what are those magic tokens and how can LuaTeX shed light on this?

In this presentation I will show examples of how LuaTeX turns characters into tokens. We will also pay some attention to the (un)usefulness of this.

Have no fear, MEGAPOST is here!

Taco Hoekwater

Anyone who has done serious work with MetaPost knows that it has quite a few implementation limits. It is not uncommon for moderately complicated graphics to occupy more than the amount of available internal memory, or to have macros that nest so deep that the stack is not large enough to hold them all. Also, values cannot be larger than 4095 without extra care.

MEGAPOST will alleviate these problems by a combination of dynamic data structure reallocation strategies and the use of a bigger internal storage type for numeric values.

DocScape Publisher: A large-scale project based on TeX

David Kastrup

The DocScape Publisher from QuinScape GmbH is focused on data-based publishing of input in XML form. At its core, currently L^ATeX, David Carlisle's `xmltex`, and pdfTeX are employed extensively. Current applications are the printing of financial reports and of a variety of product catalogs and online excerpts. Some of the problems occurring in large-scale, high-quality printing processes in the connection with TeX are explained, and solutions and products are shown.

Making of the TeX Collection

Manfred Lotz

In 1999, DANTE in collaboration with Lehmanns bookshop first produced a CTAN snapshot consisting of 3 CDs. In 2000 the TeX Live 5c CD-ROM was added and by 2002 the CTAN snapshot already consisted of 4 CDs.

After that, it was decided to produce a double layer DVD-9 consisting of a CTAN snapshot and the TeX Live 'live' image. The first DVD-9 was produced in 2003. In the following years the contents of the DVD was expanded to contain also ProTeXt, MacTeX, and ConTeXt.

This talk gives an overview about the problems we encountered when producing the first DVD-9 in 2003 and shows what steps were involved in creating the DVD.

Sanskrit typesetting from a user's perspective

Manfred Lotz

Sanskrit is an ancient Indian language, whose meaning to India is comparable to what Greek and Latin mean to European languages.

Sanskrit typesetting is very complicated, due to the existence of over 800 required ligatures. It will be shown what options are available to typeset Sanskrit under L^ATeX. The article focuses on the use of the packages `skt` and `devnag`. Their strengths and weaknesses will be discussed and examples will be given to enable the reader to get started easily.

Advanced mathematics features, for PDF and the Web

Ross Moore

Modern L^ATeX systems, creating PDF documents, support navigational features that can be usefully exploited to make technical documents much more usable than just an online facsimile of a traditional printed document. In this talk I will show various features that were developed specifically for an online version of a mathematics journal. These features include:

- metadata attachments to the PDF document;
- bookmarks to all (sub-)sections, figures, tables, theorems, and cross-referenced equations, etc., with use of Unicode strings for bookmarks, including the (simple) mathematical expressions that occur within section titles;
- searchability and copy/paste of mathematical expressions where the PDF browser recognises and supports embedded CMAP resources for the standard (e.g., CM and AMS) math fonts;
- draggable pop-ups of floating figures and tables;
- semi-automatic generation of hyperlinks to MathSciNet for bibliographic entries; i.e., helping build the *Reference Web*.

These features are all implementable now using pdfTeX; many work also with other PDF-aware drivers.

Links to documents using them can be found via the web pages at:

<http://www.austms.org.au/Publ/Bulletin/V72P1/>

<http://www.austms.org.au/Publ/Bulletin/V72P2/>

<http://www.austms.org.au/Publ/Bulletin/V72P3/>

Also, with the X_YTeX implementation of TeX, there is now direct support for OpenType fonts, and the possibility of typesetting both text and mathematics from a single font. The STIX fonts will be available soon. I will also show the results of work done by Will Robertson and myself, with criticism and advice from Chris Rowley, building upon the work of Jonathan Kew in extending X_YTeX to support mathematics. This could well become the basis of L^ATeX support for mathematics in the future.

Data structures in TeX

Marek Ryćko

The foundation of a programming language is support for data structures and operations to be performed on them. TeX, as a programming language, lacks most of the data structures known from other languages. I show how to design some basic data structures with appropriate operations and how to implement them in TeX's language in a very simple and efficient way. One of the structures introduced is a list of atomic elements, where atomic elements are TeX's token sequences. This uniform and clean way of using lists of elements makes TeX's programming much simpler and TeX's programs (macros) much more readable.

Polishing typesetting blocks

Marek Ryćko

It is now year 2007, 30 years since Donald Knuth started to implement TeX. During those 30 years thousands of programs, packages, styles, formats, fonts, scripts have been implemented in various languages, that support the "TeX way of thinking" about typesetting. We have a huge pool of programs that are capable of realizing lots of important tasks. But there is often a problem with connecting various programs together to easily achieve more complicated, structured goals.

TeX's approach to typesetting is essentially the possibility of using TeX and other related programs as links in chains or building blocks in higher level constructions. But parts of programs would also be very useful as building blocks. For example TeX's hyphenation algorithm would be very useful in many applications, not just in TeX itself. Similarly, the METAFONT and MetaPost algorithms (by John Hobby) for Bezier curve interpolation might be used in arbitrary 2D graphics applications. The monolithic constructs like TeX or MetaPost contain inside lots of programming pearls, but the pieces cannot be used separately. We have perfect building blocks of various kinds, but still are unable to build pyramids. Hopefully, after some polishing of the blocks and also after cutting some monoliths into smaller pieces, the task can be achieved.

Designing graphical signs and logotypes

Andrzej Tomaszewski

Participants in this workshop will have a chance to measure themselves designing graphical signs and/or logotypes.

ConTeXt basics for users: Table macros II

Aditya Mahajan
University of Michigan
adityam (at) umich dot edu

Abstract

This article explains some of the basic features of `table` macros in ConTeXt.

1 Introduction

In the last article, I presented some basic features of the `table` macros in ConTeXt. In this article I will present additional features of these macros. These two articles cover the most frequently used features. There are other hooks for more advanced tweaking; some are covered in the ConTeXt beginner's manual [2]; others require reading `core-tab.tex` [3]. A future article in this series may touch upon those features.

2 Specifying font style and color of columns

Sometimes you want the entire column to be set in a particular font. For example, suppose we want to produce the following table:

Year	Production (in 1,000 units)
1990	20
1991	50

One way to do this is to mark the first column in each row by `\bf`. This is time consuming and difficult to change. ConTeXt tables support an `f` key that can be used in the table preamble to set the font for the entire column. The preamble of the previous example was

```
\starttable[|f{\bf}|c|]
```

Here `f{\bf}` tells ConTeXt to typeset the first column of the table with font style `\bf`. You can use any font style command with the `f` key. Some of the more frequently used font commands have been given a key of their own. These are:

B	Bold	equivalent to <code>f{\bf}</code>
I	Italic	equivalent to <code>f{\it}</code>
S	Slanted	equivalent to <code>f{\sl}</code>
R	Roman	equivalent to <code>f{\rm}</code>
T	Teletype	equivalent to <code>f{\tt}</code>

So, we could also have written the preamble of the previous example as `\starttable[|1B|c|]`.

3 Changing the formatting of a cell

In some tables, the header (first row of the table)

needs to be bold and center aligned, while the rest of the rows are left aligned. For example

Name	Position
Someone	An important person
Someone else	A really important person

The input for this table is:

```
\starttable[|1|1|]  
  \NC \REF[cB]{...} \NC \REF[cB]{...} \NC \AR  
  \HL  
  \NC ... \NC ... \NC \AR  
  \NC ... \NC ... \NC \AR  
\stoptable
```

Notice that the table preamble says `|1|1|`, that is, both columns should be left aligned. In the first row we say `\REF[cB]{...}`, which changes the formatting of that cell to `cB`, that is, center aligned and bold. `\REF` is a short form of `\ReFormat`; both macros can be used for changing the format of the current cell. The general syntax of the command is `\REF[keys]{column content}` where `keys` can be any of the valid formatting keys accepted in the table preamble.

To change only the alignment of the current cell, you can use `\JustCenter`, `\JustLeft`, and `\JustRight`, which stand for ‘justify center’, ‘justify left’, and ‘justify right’, respectively.¹

4 Columns containing math

Suppose we want an entire column to be in math mode. For example,

Constant	Series	Value
π	$3 \sum_{n=0}^{\infty} \frac{(2n)!}{n!^2(2n+1)16^n}$	3.1415926...
e	$\sum_{n=0}^{\infty} \frac{1}{n!}$	2.7182818...

In this case, the first two columns are in math mode

¹ Most macros in ConTeXt use the word ‘align’. These macros come from the `TABLE` package, which uses the word ‘justify’.

(The second is actually in display math mode). We can manually surround each entry by $\$$; however, ConTeXt provides two ‘math column’ keys: `m` sets the column in inline math mode, and `M` sets the column in display math mode. The above example was thus keyed in as

```
\starttable[|cm|cm|l|]
  \NC \REF[c]{Constant} \NC \REF[c]{Series}
  \NC \REF[c]{Value} \NC \AR
  \HL
  \NC \pi \NC 3 \sum_{n=0}^{\infty}
    \frac {(2n)!} {n!^2 (2n+1) 16^n}
  \NC 3.1415926\dots \NC \AR
  %
  \NC e \NC \sum_{n=0}^{\infty}
    \frac 1{n!}
  \NC 2.7182818\dots \NC \AR
\stoptable
```

The first column is in inline math mode (`m` key), and the second column is in display math mode (`M` key). Notice that I have used `\REF[c]{...}` in the first row, so the headings are not in math mode.

5 Numeric columns

Tables containing statistical data need the data to be aligned at the decimal place. ConTeXt provides two keys for this: `n` displays the column in text mode, while `N` displays it in math mode. Both keys take a space-delimited argument of the form `x.y` where `x` is the number of digits before the decimal and `y` is the number of digits after the decimal. For example, to get the following table (adapted from Tobias Oetiker’s “The not so short introduction to L^AT_EX”):

Pi expression	Value
π	3.1416
π^π	36.46
$(\pi^\pi)^\pi$	80,662.7

I keyed in

```
\starttable[|cm|n5.4 |]
  \NC \REF[c]{Pi expression}
  \NC \REF[c]{Value} \NC \AR
  \HL
  \NC \pi \NC 3.1416 \NC \AR
  \NC \pi^{\pi} \NC 36.46 \NC \AR
  \NC (\pi^{\pi})^{\pi} \NC 80,662.7 \NC \AR
  \HL
\stoptable
```

The key `|n5.4 |` (notice the space at the end) means that we want five digits before the decimal and four digits after the decimal.

Some European countries use a comma as a decimal separator. This can be done using the `q` and `Q`

keys. They take a space-delimited argument of the form `x,y` which has the same meaning as the argument of `n` and `N` keys. So, to get this table

Pi expression	Value
π	3,1416
π^π	36,46
$(\pi^\pi)^\pi$	80.662,7

I keyed in

```
\starttable[|cm|q5,4 |]
  \NC \REF[c]{Pi expression}
  \NC \REF[c]{Value} \NC \AR
  \HL
  \NC \pi \NC 3,1416 \NC \AR
  \NC \pi^{\pi} \NC 36,46 \NC \AR
  \NC (\pi^{\pi})^{\pi} \NC 80.662,7 \NC \AR
  \HL
\stoptable
```

An `n` or `N` column must contain a dot; a `q` or `Q` column must contain a comma. For cells that do not contain a dot or comma (for example, the headings of the table) we can use `\REF` to change the formatting. The TeX primitive `\omit` can be used to leave the cell empty.

6 Spanning multiple columns and rows

In table heads, one often needs a cell spanning multiple columns. ConTeXt provides the `\use` macro to do this. This macro takes an argument specifying the number of columns to span. For example, to use five columns, we can use `\use{5}`. The macros `\TWO`, `\THREE`, `\FOUR`, `\FIVE`, `\SIX` are shortcuts to span the corresponding number of columns.

By default, when spanning multiple columns, the formatting keys of the last spanned column are in effect. We can use `\REF` to change the formatting. ConTeXt also provides `\Use` (note the uppercase `U`) to span multiple columns and also set the formatting: to span three columns and make the content center aligned we can use `\Use{3}[c]{content}`.

The support for spanning multiple rows is more limited. There are two commands `\Lower` and `\Raise` that can lower or raise the contents of the cell. There are two forms of these commands: `\Raise{5}{content}` which raises the content by 5 times 1/12th of the line height; and `\Raise(dimen){content}` which raises the content by `dimen` (which can be any valid TeX dimension).

The most common usage of spanning multiple rows is spanning two rows in table heads. For that, we can use `\LOW`, which lowers the current cell by half of the line height, making it visually centered between the two rows.

Here is a table showing both column and row spanning (example adapted from Andy Roberts' L^AT_EX tutorial [4]):

Team Sheet		
Goal Keeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duberry
	DC	Dominic Matteo
	RB	Didier Domi

This was typed as

```
\starttable[|1|1|1|]
\HL
\VL \Use{3}[c]{Team Sheet} \VL \AR
\HL
\VL Goal Keeper \VL GK
\VL Paul Robinson \VL \AR
\HL
\VL \Lower(1.5\lineheight){Defenders} \VL
\VL LB \VL Lucus Radebe \VL \AR \VL \VL
\VL DC \VL Michael Duberry \VL \AR \VL \VL
\VL DC \VL Dominic Matteo \VL \AR \VL \VL
\VL RB \VL Didier Domi \VL \AR
\HL
\stoptable
```

In the first row we use `\Use{3}[c]{...}` to span three columns and make the cell center aligned. In the first column of the third row, we use `\Lower(1.5\lineheight){...}` to lower the cell so that it appears to be visually centered in the last four rows.

7 Controlling space between columns

By default, there is a 0.5em (usually about half the current font size) space between the columns. We can change this using the `o` and the `s` keys. The `o` key changes the space on the right of the current column; the `s` key changes the space of all the following columns until the next `o` or `s` key.

There are two ways of specifying the parameters of the `o` and `s` keys. The first is in integer multiples of 0.5em: `s{n}` makes the space equal to `n` times 0.5em. So, to get a space of 1.5em between columns we can use `\starttable[s{3}|1|1|]`. The second way is to specify the distance as an arbitrary T_EX dimension. So, we could also have used `\starttable[s(1.5em)|1|1|]`. Notice that in the first case, the argument is given in curly brackets;² in the second, the argument is given in parentheses.

It is also possible to add padding (kerning) to the left and/or right of each column. The key `i` adds padding to the left, `j` adds padding to the right, and

`k` adds padding to both the right and the left.³ The amount of padding is specified in the same way as in the `o` and the `s` keys: either in multiples of 0.5em, or as arbitrary dimensions.

A combination of these keys can be used to force the `table` macros to produce tables as recommended by the `booktabs` package [1]. For example

```
\setuptables[rulethickness=0.03em]
\starttable[s0|i1|i2|i2r|]
\HL[3]
\NC \Use2[c]{Item} \NC \NC \AR
\DL[2] \DC \DR
\NC Animal \NC Description \NC Price (\$)\NC \AR
\HL[2]
\NC Gnat \NC per gram \NC 13.65 \NC \AR
\NC \NC each \NC 0.01 \NC \AR
\NC Gnu \NC stuffed \NC 92.50 \NC \AR
\NC Emu \NC stuffed \NC 33.33 \NC \AR
\HL[3]
\stoptable
```

gives

Item		
Animal	Description	Price (\$)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33

Notice that in this case, the horizontal lines do not extend beyond the table. The half line⁴ after the first row extends only until the end of the second column. Compare this with the table that we get from `\starttable[|1|1|r|]`:

Item		
Animal	Description	Price (\$)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33

We will not go into the details of coaxing and beating `table` macros into producing tables like the `booktabs` package: they were never designed for that task. We can achieve the simpler parts of

² Actually, this is one argument according to T_EX's parsing rule. So, for single digit arguments, we can omit the curly brackets.

³ We are really running out of letters of the alphabet!

⁴ In these articles, I have only talked about `\HL` and not explained how to get *division lines* between rows. This is explained in the ConTeXt manual [2].

the `booktabs` recommendation, but for more complicated things such as `\cmidrule`, the `table` macros do not have enough hooks.

To achieve lines that get trimmed at the edge of the table, we can use `\starttable[o0|1|1|ro0|]`,⁵ which gives:

Item		
Animal	Description	Price (\$)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33

The division line in this case extends to the middle of the second column. If the table does not have division lines, adding `o0` in the beginning and the end of the table preamble is usually sufficient.

8 Controlling space between rows

The `table` macros do not provide much control over space between rows of the table. You can have loose or tight tables by changing the `distance` option of `\setuptables`. The `distance` option takes four values: `none`, `small`, `medium`, and `big`. The default is `medium`. For example, let's reconsider the table of Section 6. With `\setuptables[distance=none]`, we get

Team Sheet		
Goal Keeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duberry
	DC	Dominic Matteo
	RB	Didier Domi

while with `\setuptables[distance=big]` we get

Team Sheet		
Goal Keeper	GK	Paul Robinson
Defenders	LB	Lucus Radebe
	DC	Michael Duberry
	DC	Dominic Matteo
	RB	Didier Domi

Play around with these values to find out what value of `distance` you prefer. It is possible to get more control using the `\OpenUp` macro of the `TABLE` package, but there is no interface for that. See the discussion on the mailing list [5] for an example.

9 Remembering preambles

Often one has several tables which need to have similar formatting. Repeating the table preamble in each case is error-prone. `ConTeXt` provides `\definetabletemplate` which can be used to specify a table preamble which can be reused later. For example, we can say

```
\definetabletemplate[booktabs][o0|1|1|ro0]
```

Then we can invoke this preamble by

```
\starttable[booktabs]
```

10 Other features

When I started writing these articles on the `table` macros, I thought that one article would be enough. About halfway through the first article I realized that I would need more than one article. Now I find that even two are not enough. There are lots of things that I have not even touched; using color in tables and breaking the table across pages are the most important omissions. These will have to wait for a later article in this series. Next issue, we will look at something different.

11 References

- [1] Simon Fear and Danie Els, “Publication quality tables in L^AT_EX”, <http://www.ctan.org/tex-archive/macros/latex/contrib/booktabs>
- [2] Hans Hagen, `ConTeXt`: an excursion. <http://www.pragma-ade.com/show-man-1.htm>
- [3] Hans Hagen, `ConTeXt` core macros — `TABLE` embedding. http://www.logosrl.it/context/modules/current/singles/core-tab_ebook.pdf
- [4] Andy Roberts, L^AT_EX Tutorials — Tables. <http://www.andy-roberts.net/misc/latex/latextutorial4.html>
- [5] Discussion on the `ConTeXt` mailing list, <http://archive.contextgarden.net/message/20070806.011325.5a938ae7.en.html>

⁵ Wait a minute! To which column does the first `o0` correspond? The table consists of a virtual column at the left edge, typically for drawing a vertical line there. The key `o0` in the beginning of the preamble sets the width of this virtual column to be zero.

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>.

Kinch, Richard J.

7890 Pebble Beach Ct.
Lake Worth, FL 33467
+1 561 966-8400
Email: [kinch \(at\) truetex.com](mailto:kinch@truetex.com)

Publishes TrueTeX, a commercial implementation of TeX and LaTeX. Custom development for TeX-related software and fonts.

Martinez, Mercè Aicart

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) menta.net](mailto:m.aicart@menta.net)
Web: www.edilatex.com/

We provide, at reasonable low cost, TeX and LaTeX typesetting services to authors or publishers worldwide. We have been in business since the beginning of 1990. For more information visit our web site.

TUG Institutional Members

Aalborg University, Department of Mathematical Sciences,
Aalborg, Denmark

American Mathematical Society,
Providence, Rhode Island

Banca d'Italia, *Roma, Italy*

Center for Computing Sciences,
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, Czech Republic

Moravian College, Department of Mathematics and Computer Science,
Bethlehem, Pennsylvania

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

University College, Cork,
Computer Centre, *Cork, Ireland*

University of Delaware,
Computing and Network Services,
Newark, Delaware

Université Laval,
Ste-Foy, Québec, Canada

Universiti Tun Hussein Onn Malaysia,
Pusat Teknologi Maklumat,
Batu Pahat, Johor, Malaysia

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

Vanderbilt University,
Nashville, Tennessee

Peter, Steve

310 Hana Road
Edison, NJ 08817
+1 732 287-5392
Email: [speter \(at\) dandy.net](mailto:speter@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
+1 703 860-0013
Email: [borisv \(at\) lk.net](mailto:borisv@lk.net)
Web: <http://borisv.lk.net>

TeX and LaTeX consulting, training and seminars. Integration with databases, automated document preparation, custom LaTeX packages, conversions and much more. I have about twelve years of experience in TeX and twenty-five years of experience in teaching & training. I have authored several packages on CTAN and published papers in TeX related journals.

In memoriam Bernard Gaulle

Maurice Laugier*

The past president and founder of GUTenberg¹ passed away on 2nd August. Here I would like to pay a tribute to him. As our association was unable to be represented at his funeral, we sent a telegram.

Bernard Gaulle was quite discreet, even secret about himself: it has been difficult for us to reconstitute his career. First he was a computer scientist at the Blaise Pascal Institute, part of the CNRS,² grouping scientific and management applications at the end of the 1960s. Then, as soon as the national centre of scientific computer engineering (CIRCÉ³) was created at Orsay in 1969, he entered it as a system engineer specialised in operating systems like MFT,⁴ ASP, MVS⁵-JESS3, on the IBM⁶ mainframe. He became the head of the user assistance group and stayed on at this position until 1993. In particular, he was in charge of publications (users' documentation, course manuals, newsletters) and was interested in the rise of DTP⁷ tools. In 1983–84, 'a researcher turned up in my office, carrying a magnetic tape that came from Stanford,' he narrated in No. 0 of the *GUTenberg Letter*, dated February 1993. The history of the beginning of the GUTenberg association can also be found in this issue. That was how he discovered T_EX, was persuaded, and spent his life to promote it not only for scientists — which was quite easy — but also the general public.

About 1984, he set up the SAPRISTI⁸ system at CIRCÉ. He conceived it out of T_EX, developed it, and got his secretaries to use it. At this time, that was very innovative and . . . audacious!⁹

In June 1984, he met Jacques André and together they launched a project for a French-speaking T_EX Users Group. Four years were needed in order for this association to be set up. However, Bernard Gaulle was already organising manifestations and represented French-speaking people at the first

EuroT_EX conference at Como, Italy. The GUTenberg association was created 23rd September 1988, and Bernard Gaulle was elected President.

He was actively involved in the association and was re-elected President. In this way, he participated in TUG's board of directors as Vice-President for GUTenberg, then as Special Director for GUTenberg. He regularly published editorials and notes concerning his ideas about T_EX and the association. To get more involved in his own work and his programs outside T_EX, he resigned his position as President in 1992 and passed the office to Alain Cousquer.

Another turning point in his career happened in 1993: CIRCÉ was replaced by IDRIS.¹⁰ Bernard was a *chargé de mission* by IDRIS' Director for the distributed systems for scientific computer engineering.

Bernard Gaulle devoted his efforts to the maintenance and diffusion of his **french** package since 1993. His numerous articles in the *Cahiers GUTenberg* and *GUTenberg Letter* informed the French-speaking T_EX community.

At the same time, he was also interested in French legal problems regarding software put on the Internet (he presented a survey about that in *Cahiers GUTenberg*, No. 25). This led him to create the *Litiel*¹¹ association. He became President of this association and was involved in it until his passing, although he had already left IDRIS in May 2005 for health reasons: a cancer he faced bravely.

Bernard did much for both GUTenberg and T_EX in general. He left not only two major works (GUTenberg and **french**) but also a recollection of a kind president, pleasant, and devoted. He was very active, resolute in his developments, even pugnacious.

We wish to express our pain and sympathy to his wife, Catherine Gaulle — who participated in preparing GUTenberg and EuroT_EX meetings — and their daughters.

(English translation: Jean-Michel Hufflen.

The original appeared in *La lettre GUTenberg*, numéro 34, octobre 2007.)



¹⁰ *Institut du Développement et des Ressources en Informatique Scientifique*. Institute of development and resources for scientific computer engineering.

¹¹ From 'logiciel', French for 'software'.

* Current president of GUTenberg.

¹ *Groupe francophone des Utilisateurs de T_EX*, French-speaking T_EX users group.

² *Centre National de la Recherche Scientifique*. French governmental organisation of scientific research.

³ *Centre InterRégional de Calcul Électronique*, inter-regional centre for electronic computation.

⁴ Multiprogramming with a Fixed Number of Tasks.

⁵ Multiple Virtual Storage.

⁶ International Business Machines.

⁷ DeskTop Publishing.

⁸ *Système Assisté de P_Roduction Intégrant Simplement Textes et Images*, literally: 'assisted production system integrating texts and images easily.'

⁹ Bernard Gaulle narrated this part of his life and his problems at this time in two articles of the French group's journal (*Cahiers GUTenberg*): the first is included in No. 0 (1988), the second in Nos. 15 and 17 (1993).

EuroBachTeX 2007

TUGboat, Volume 29, Number 1, 2008

Buletyn GUST, Zeszyt 24, 2007

Die TeXnische Komödie, 20. Jahrgang, 1/2008



EuroBachTeX 2007	2	Conference program, delegates, and sponsors
	6	Jerzy Ludwiczowski and Petr Sojka / EuroBachTeX 2007: Paths to the Future
	13	Sam Guravage / Confessions of a teenage TeX user
Typography	14	Grażyna Jackowska / Handmade paper: A mixture of handcraft, art and fun
	16	Andrzej Tomaszewski / Designing a special book: With both pleasure and ... fear
	20	Dorota Cendrowska / Enumerations as an interesting form of text appearance
Fonts	25	Jerzy Ludwiczowski, Bogusław Jackowski and Janusz Nowacki / Five years after: Report on international TeX font projects
	27	Janusz Nowacki / Cyklop: A new font family
	28	Hans Hagen / Do we need a font system in TeX?
	34	Taco Hoekwater / OpenType fonts in LuaTeX
	36	Hàn Thê Thành / Font-specific issues in pdfTeX
	42	Karel Horák / Those obscure accents ...
	45	Klaus Höppner / Creation of a PostScript Type 1 logo font with MetaType 1
	50	Karel Píška / Procedures for font comparison
	57	Karel Píška / Comments and suggestions about the Latin Modern fonts
	66	Jerzy Ludwiczowski and Karl Berry / The GUST Font License: An application of the L ^A TeX Project Public License
Resources	68	Arthur Reutenauer / A brief history of TeX, volume II
	73	Ulrik Vieth / Overview of the TeX historic archive
	77	Joanna Ludmiła Ryćko / TeX Clinic
Multilingual Document Processing	79	Ameer Sherif and Hossam Fahmy / Parameterized Arabic font development for AlQalam
	89	Atif Gulzar and Shafiq ur Rahman / Nastaleeq: A challenge accepted by Omega
	95	Hàn Thê Thành / Typesetting Vietnamese with VnTeX (and with the TeX Gyre fonts too)
	101	Jean-Michel Hufflen / Managing order relations in MIBIBTeX
Electronic Documents	109	Jean-Michel Hufflen / Introducing L ^A TeX users to XSL-FO
	117	Tomasz Łuczak / Using TeX in a wiki
Publishing	118	Petr Sojka and Michal Růžička / Single-source publishing in multiple formats for different output devices
	125	Péter Szabó / Practical journal and proceedings publication on paper and on the web
Software & Tools	133	Jim Hefferon / An experimental CTAN upload process
	136	Norbert Preining / TeX (Live) on Debian
	140	Siep Kroonenberg / Epspdf: Easy conversion between PostScript and PDF
	143	Martin Schröder / pdfTeX 1.40: What's new
	146	Jonathan Kew / XeTeX Live
	151	Gerd Neugebauer / Conventional scoping of registers—An experiment in ε _X TeX
	157	Jean-Michel Hufflen / MIBIBTeX: Reporting the experience
	163	David Kastrup / Writing (L ^A)TeX documents with AUCTeX in Emacs
	164	Tomasz Łuczak / LyX: An editor not just for secretaries
	166	Péter Szabó / Automated DVD menu authoring with pdfL ^A TeX
L^ATeX	176	Zofia Walczak / Graphics in L ^A TeX using TikZ
	180	Grzegorz Murzynowski / L ^A TeX vs. L ^A TeX—a modification of the logo
	181	David Kastrup / Benefits, care and feeding of the bigfoot package
	184	Johannes Große / MathPSfrag: L ^A TeX labels in Mathematica plots
	190	David Kastrup / makematch, a L ^A TeX package for pattern matching with wildcards
	193	David Kastrup / qstest, a L ^A TeX package for unit tests
Macros	199	Grzegorz Murzynowski / gmverse and gmcontinuo—some nontrivial placement of text on a page
	201	Grzegorz Murzynowski / The gmdoc bundle—a new tool for documenting (L ^A)TeX sources
Hints & Tricks	207	Paweł Jackowski / TeX beauties and oddities: A permanent call for TeX pearls
Puzzle	216	Janusz Nowacki / Crossword
Abstracts	217	Abstracts (Hagen, Hoekwater, Kastrup, Lotz, Moore, Ryćko, Tomaszewski)
ConTeXt	219	Aditya Mahajan / ConTeXt basics for users: Table macros II
TUG Business	223	TUG institutional members
Advertisements	223	TeX consulting and production services
Memorial	224	Maurice Laugier / In memoriam Bernard Gaulle

Table of Contents (ordered by difficulty)**Introductory**

- 20 *Dorota Cendrowska* / Enumerations as an interesting form of text appearance
 • classical and modern formatting of enumerations
- 13 *Sam Guravage* / Confessions of a teenage T_EX user
 • report on the talk, with photos
- 42 *Karel Horák* / Those obscure accents . . .
 • discussion and comparison of the many versions of caron (háček) accents
- 14 *Grażyna Jackowska* / Handmade paper: A mixture of handcraft, art and fun
 • report on a papermaking workshop, with photos
- 163 *David Kastrup* / Writing (L^A)T_EX documents with AUCT_EX in Emacs
 • introduction to the AUCT_EX Emacs facilities, and download locations
- 146 *Jonathan Kew* / X_YT_EX Live
 • X_YT_EX's incorporation in T_EX Live 2007, and further developments
- 140 *Siep Kroonenberg* / Epspdf: Easy conversion between PostScript and PDF
 • command-line and GUI interface to convenient graphics conversion
- 164 *Tomasz Łuczak* / L_YX: An editor not just for secretaries
 • overview of L_YX, an editor for T_EX with a graphical interface
- 117 *Tomasz Łuczak* / Using T_EX in a wiki
 • overview of using T_EX as a back-end for PDF production from a wiki
- 66 *Jerzy Ludwichowski* and *Karl Berry* / The GUST Font License: An application of the L^AT_EX Project Public License
 • using the LPPL (with an additional request) for fonts
- 25 *Jerzy Ludwichowski*, *Bogusław Jackowski* and *Janusz Nowacki* / Five years after: Report on international T_EX font projects
 • status and samples of these two major font projects
- 6 *Jerzy Ludwichowski* and *Petr Sojka* / EuroBachoT_EX 2007: Paths to the Future
 • introduction to the conference and proceedings
- 180 *Grzegorz Murzynowski* / L^AT_EX vs. L^AT_EX — a modification of the logo
 • a L^AT_EX logo definition that self-adjusts to many fonts
- 27 *Janusz Nowacki* / Cyklop: A new font family
 • a heavy sans serif oblique for titling and displays
- 68 *Arthur Reutenauer* / A brief history of T_EX, volume II
 • recapitulation of T_EX origins, evolution, and descendants
- 77 *Joanna Ludmiła Ryćko* / T_EX Clinic
 • overview of the T_EX help clinic available at Bachotek and via email
- 16 *Andrzej Tomaszewski* / Designing a special book: With both pleasure and . . . fear
 • creation of a commemorative edition for the Warsaw Waterworks jubilee
- 73 *Ulrik Vieth* / Overview of the T_EX historic archive
 • preserving T_EX distributions and packages for software archaeologists
- 176 *Zofia Walczak* / Graphics in L^AT_EX using TikZ
 • introduction to TikZ for drawing graphics directly in L^AT_EX

Intermediate

- 28 *Hans Hagen* / Do we need a font system in T_EX?
 • reflections on the font system in T_EX generally, ConT_EXt specifically, and coming changes
- 36 *Hàn Thế Thành* / Font-specific issues in pdfT_EX
 • microtypography, letterspacing, interword spacing, character kerning, subfont, runpdfTeX
- 95 *Hàn Thế Thành* / Typesetting Vietnamese with VnT_EX (and with the T_EX Gyre fonts too)
 • introduction to typesetting Vietnamese in T_EX
- 133 *Jim Hefferon* / An experimental CTAN upload process
 • a cooperative web-based CTAN (and T_EX Live) package processing method
- 34 *Taco Hoekwater* / OpenType fonts in LuaT_EX
 • overview of the state of reading OpenType directly from LuaT_EX

- 45 *Klaus H \ddot{o} ppner* / Creation of a PostScript Type 1 logo font with MetaType 1
 - tutorial for implementing a font with MetaType 1
- 109 *Jean-Michel H \ddot{u} fflen* / Introducing L^AT_EX users to XSL-FO
 - concise introduction to XSL-FO with comparisons to L^AT_EX
- 181 *David Kastrup* / Benefits, care and feeding of the **bigfoot** package
 - improved page breaking and other footnote enhancements
- 219 *Aditya Mahajan* / ConT_EXt basics for users: Table macros II
 - further overview of making tables in ConT_EXt
- 199 *Grzegorz Murzynowski* / **gmverse** and **gmcontinuo** — some nontrivial placement of text on a page
 - optical centering and right alignment of verses; continuous paragraph setting
- 136 *Norbert Preining* / T_EX (Live) on Debian
 - usage of T_EX on Debian, including package and font installation
- 143 *Martin Schröder* / pdfT_EX 1.40: What's new
 - overview of new features: JBIG2, color stacks, transformation matrices, and more
- 118 *Petr Sojka* and *Michal Růžička* / Single-source publishing in multiple formats for different output devices
 - using high-level markup, pdfL^AT_EX, and T_EX4ht for multiple output formats
- 125 *Péter Szabó* / Practical journal and proceedings publication on paper and on the web
 - techniques and advice for editorial workflow and production operations

Intermediate Plus

- 184 *Johannes Große* / MathPSfrag: L^AT_EX labels in Mathematica plots
 - typographically consistent labels for Mathematica plots, using PDF or PostScript
- 89 *Atif Gulzar* and *Shafiq ur Rahman* / Nastaleeq: A challenge accepted by Omega
 - implementing the Urdu script in Omega
- 101 *Jean-Michel H \ddot{u} fflen* / Managing order relations in MIB_BT_EX
 - handling language-specific lexicographic orderings
- 157 *Jean-Michel H \ddot{u} fflen* / MIB_BT_EX: Reporting the experience
 - discussing design and implementation choices of the MIB_BT_EX program
- 201 *Grzegorz Murzynowski* / The **gmdoc** bundle — a new tool for documenting (L^A)T_EX sources
 - an enhanced reimplement of **doc**
- 50 *Karel Piška* / Procedures for font comparison
 - tools and examples for detailed comparison of glyphs, kerns, and more
- 57 *Karel Piška* / Comments and suggestions about the Latin Modern fonts
 - comparisons among Latin Modern, Computer Modern, and the Czech/Slovak CS fonts
- 79 *Ameer Sherif* and *Hossam Fahmy* / Parameterized Arabic font development for AlQalam
 - achieving high-quality Arabic typesetting through METAFONT pens and drawing
- 166 *Péter Szabó* / Automated DVD menu authoring with pdfL^AT_EX
 - creating standard DVDs with menus created in L^AT_EX

Advanced

- 207 *Paweł Jackowski* / T_EX beauties and oddities
 - the 2007 edition of T_EX pearls
- 190 *David Kastrup* / **makematch**, a L^AT_EX package for pattern matching with wildcards
 - efficient pattern matching supporting * and !
- 193 *David Kastrup* / **qstest**, a L^AT_EX package for unit tests
 - extensive unit testing functionality, including embedding in **dtx** files
- 151 *Gerd Neugebauer* / Conventional scoping of registers — An experiment in $\epsilon\lambda$ T_EX
 - alternative localization of registers via the Java infrastructure of $\epsilon\lambda$ T_EX
- 216 *Janusz Nowacki* / Crossword

Reports and notices

- 224 *Maurice Laugier* / *In memoriam* Bernard Gaulle
 - 2 EuroT_EX 2007 conference delegates and sponsors
 - 4 EuroT_EX 2007 program
 - 8 EuroT_EX 2007 photos
- 217 *Abstracts* (Hagen, Hoekwater, Kastrup, Lotz, Moore, Ryćko, Tomaszewski)
- 223 Institutional members
- 223 T_EX consulting and production services