

Zpravodaj Československého sdružení uživatelů T_EXu Zpr...

sdružení uživatelů T_EXu Zpravodaj Československého sdružení už...

j Československého sdružení uživatelů T_EXu Zpravodaj Českoslo...

elů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zp...

ého sdružení uživatelů T_EXu Zpravodaj Československého sdružen...

vodaj Československého sdružení uživatelů T_EXu Zpravodaj Česko...

živatelů T_EXu Zpravodaj Československého sdružení uživatelů T_E...

venského sdružení uživatelů T_EXu Zpravodaj Československého s...

Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj C...

ení uživatelů T_EXu Zpravodaj Československého sdružení uživate...

skoslovenského sdružení uživatelů T_EXu Zpravodaj Československ...

T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpra...

sdružení uživatelů T_EXu Zpravodaj Československého sdružení už...

j Československého sdružení uživatelů T_EXu Zpravodaj Českoslov...

elů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zp...

ého sdružení uživatelů T_EXu Zpravodaj Československého sdružen...

vodaj Československého sdružení uživatelů T_EXu Zpravodaj Česko...

živatelů T_EXu Zpravodaj Československého sdružení uživatelů T_E...

venského sdružení uživatelů T_EXu Zpravodaj Československého s...

Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj C...

ení uživatelů T_EXu Zpravodaj Československého sdružení uživate...

skoslovenského sdružení uživatelů T_EXu Zpravodaj Československ...

T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpra...

sdružení uživatelů T_EXu Zpravodaj Československého sdružení už...

j Československého sdružení uživatelů T_EXu Zpravodaj Českoslov...

elů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zp...

ého sdružení uživatelů T_EXu Zpravodaj Československého sdružen...

vodaj Československého sdružení uživatelů T_EXu Zpravodaj Česko...

živatelů T_EXu Zpravodaj Československého sdružení uživatelů T_E...

venského sdružení uživatelů T_EXu Zpravodaj Československého s...

Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj C...



ZPRAVODAJ

ení uživatelů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj Č...

skoslovenského sdružení uživatelů T_EXu Zpravodaj Československého sdružení uživatelů

Československého sdružení uživatelů T_EXu

elů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj Českoslov...

ského sdružení uživatelů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpra...

vodaj Československého sdružení uživatelů T_EXu Zpravodaj Československého sdružení u...

živatelů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj Českoslo...

venského sdružení uživatelů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpra...

Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj Československého sdruž...

ení uživatelů T_EXu Zpravodaj Československého sdružení uživatelů T_EXu Zpravodaj Č...

2-4

MAPS 43

2011

Zpravodaj Československého sdružení uživatelů \TeX u je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes <http://www.cstug.cz/>.

Zpravodaj je zařazen do Seznamu recenzovaných neimpaktovaných periodik vydávaných v České republice, viz <http://www.vyzkum.cz/>.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip`, `.arj`, `.tar.gz`). Postupujte podle instrukcí, které najdete na stránce <http://bulletin.cstug.cz/>. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě, CD, či DVD na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3
zpravodaj@cstug.cz

Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí \TeX Live), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)

ISSN 1213-8185 (online verze)





Photos by Frans Goddijn – IX/2010

Conference Proceedings of
The Fourth International CONTEXt Meeting
and The Third TEXperience Conference



CS *TU* **Ω**

K-PATENTS
PROCESS INSTRUMENTS

KORUNNÍ

**MŠ
MT**

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

 **Tomas Bata University in Zlín**
Faculty of Management and Economics

Preface

Předmluva

JÁN KULA, PAVEL STRÍŽ

Dear T_EX lovers, developers and users; dear friends, ladies and gentlemen,

You have just opened the conference proceedings from a double-conference, the fourth international C_ON_TE_XT meeting (4CM) and the third T_EXperience conference (3TE; an annual C_STUG meeting). Both took place in the Brejlov mill (Prague), the Czech Republic, www.brejlov.cz, on September 13–19, 2010.

The informal conference themes were:

C_ON_TE_XT typesetting documentation, teach as we preach, along with Plain T_EX, L^AT_EX, C_ON_TE_XT and LuaT_EX are best friends!

We are honoured that the fourth C_ON_TE_XT meeting and the third T_EXperience conference took place under the aegis of the Ministry of Education, Youth and Sports of the Czech Republic, www.msmt.cz.

We are also proud that the Czechoslovak T_EX Users Group, www.cstug.cz, and Faculty of Management and Economics of Tomas Bata University in Zlín, the Czech Republic, www.fame.utb.cz, took the patronage of conferences.

Our thanks go to sponsors and benefactors without whom the conferences would be difficult to manage and impossible to make the participants feel like home, surrounded by family and pets.

Enjoy the articles, photos and supplemental material included in the proceedings. We made as few editorial corrections as possible.

We hope we will meet you at a T_EX or Open Source conference soon. If not then see you in 2013 in Brejlov!

Sincerely Yours,

Ján Kula, Pavel Stríž
On behalf of the organizers

4th ConTEXT meeting



CON
TEXT

Czech Republic, Břejčlov (13.-18. 9. 2010)



1. Hraban Ramm Henning
2. Piotr Strzelczyk
3. Nino Bašić
4. Michael Guravage
5. Mari Voipio

6. Lubor Homolka
7. Wolfgang Schuster
8. Steffen Wolfrum
9. Jano Kula
10. Patrick Gundlach

11. Ulrik Vieth
12. Mojca Miklavc
13. Arthur Reutenauer
14. Willi Eger
15. Hans Hagen

16. Luigi Scarso
17. Teco Hoekwater
18. Alan Braslau
19. Lukáš Procházka
20. Pavneet Arora

21. Milan Stříž
22. Harald König
23. Jaroslav Hajtmar
24. Karel Piška
25. Eva Van Deventer

The remaining participants were invisible ©

Photo by Frans Goddijn

Programme of ConT_EXt meeting and T_EXperience

Brejlov 13. – 19. 9. 2010

Monday 13.9.

18:30	Dinner			90	
20:00	Taco Hoekwater / Hans Hagen / Mojca Miklavc	ConTeXt installation clinic	workshop	120	

Tuesday 14.9.

8:00	Breakfast			60	
09:00		Conference opening		15	
09:15	Arthur Reutenauer	Keynote talk	talk	15	
09:30	Taco Hoekwater	Reference manual update	talk	15	
09:45	Patrick Gundlach	Wiki context reference	talk	15	
10:00	Taco Hoekwater / Hans Hagen	Other documentation news	talk	15	
10:15	Peter Münster (presented by Taco)	Reference syntax in lua	talk	15	
10:30	Coffee			30	
11:00	Hans Hagen	Why structure matters	talk	30	
11:30	Wolfgang Schuster	Module documentation	talk	30	
12:00	Hans Hagen	A Context Scoop	talk	30	
12:30	Lunch			90	
14:00	Mari Voipio	A Different Philosophy I Thoughts on teaching ConTeXt to non-techies	discussion	30	
14:30	Mari Voipio	Observations on approaches to learning ConTeXt and writing documentation to match	discussion	60	
15:30	Coffee			30	
16:00	Wolfgang Schuster	xml interface files and internal module doku (%D, %M etc.)	talk	30	
16:30	Hans Hagen	Requirements For Documentation	talk	30	
17:00	Hans Hagen	My Slowly Growing Test Suite	talk	30	
17:30		Documentation discussion	discussion	60	
18:30	Dinner			90	
20:00	Taco Hoekwater	tlcontrib.metatex.org	talk	30	
20:30	Mojca Miklavc	ConTeXt minimal, "Server edition"	talk+discussion	30	
21:00	Patrick Gundlach	contextgarden.net	q&a	30	

Wednesday 15.9.

8:00	Breakfast			60	
09:00	Mari Voipio	Uses for ConTeXt in a standard office environment	talk	30	
09:30	Hans Hagen / Mojca Miklavc	The database module, MkIV version	talk	30	
10:00	Hans Hagen	XML processing news	talk+workshop	60	
11:00	Coffee			30	
11:30	Alan Braslau	Plotting data with Metafun/Metapost	talk+discussion	45	

12:15	Alan Braslau	Drawing diagrams using the chart module	talk+discussion	45	
13:00	Lunch				
14:30	Hans Hagen	Font Goodies	talk	30	
15:00	Mojca Miklavec	Some thoughts on typescripts	talk+discussion	60	
16:00	Coffee				
16:30	Aditya Mahajan (presented by Luigi)	Beg, borrow, and steal – running external filters in ConTeXt	talk	30	
17:00	Alan Braslau	Drawing chemical structures using ppchTeX	talk	30	
17:30	Wolfgang Schuster	The letter module	talk+workshop	90	
19:00	Dinner				
20:30	Luigi Scarso	modules_mkiv	talk	30	
21:00	Hans Hagen	Document workflow	tutorial	30	
21:30	Hans Hagen	Whatever You Want To Know	q&a	90	

Thursday 16.9.

8:00	Breakfast				
09:00	Luigi Scarso	Souvenir d'Italie	talk	30	
09:30	Hans Hagen	Context Lua Documents	talk	30	
10:00	Coffee				
10:30	Taco Hoekwater	Escrito	talk	30	
11:00	Patrick Gundlach	LuaTeX without TeX – or: the hidden beauty of TeX	talk	45	
11:45	Aditya Mahajan (presented by Taco)	Math wishlist	discussion	30	
12:15	Lunch				
13:30		Relax time to be announced (trip, walk, excursion)			

Friday 17.9.

8:00	Breakfast				
09:30	Hans Hagen	How Luatex and Context Proceed	talk	30	
10:00	Taco Hoekwater	The current state of LuaTeX	talk	30	
10:30	Taco Hoekwater	Metapost developments	talk	30	
11:00	Coffee				
11:30	Arthur Reutenauer	The ever-regenerating hydra: hyphenation patterns in Unicode, and beyond	talk	30	
12:00	David Březina	General issues in multi-script typography	talk	45	
12:45	Ulrik Vieth	Experiences typesetting OpenType math with LuaLaTeX and XeLaTeX	talk	30	
13:15	Lunch				
14:45	Karel Piška	Fonts with complex OpenType tables	talk	30	
15:15	David Březina	Skolar – Designing a Typeface for Academic Publications: http://www.type-together.com/Skolar	talk	30	
15:45	Piotr Strzelczyk	Short info about new release of Antykwa Półtawskiego font	talk	30	
16:15	Coffee				
16:45	Luigi Scarso	Playing with Flash in ConTeXt-mkiv	talk	30	
17:15	Karel Horák	Do it better ...	talk	30	

17:45	Taco Hoekwater	Lua for Beginners	workshop	60	
18:45		Conference close		15	
19:00	Dinner			90	
20:30	Taco Hoekwater	Lua for Beginners	workshop	30	
21:00	Taco Hoekwater	Metapost details	demo	30	

Saturday 18.9.

8:00	Breakfast			60	
09:00	Taco Hoekwater	Lua for font lovers	workshop	90	
10:30	Coffee			30	
11:00	Willi Egger	Arranging Pages for Printing + Creating a Flyer	talk+workshop	90	
12:30	Lunch			90	
14:00	John Haltiwanger	Subtext: A Proposed Processual Grammar for a Multi-Output Pre-Format	talk	30	
14:30	Idris Samawi Hamid (presented by Taco)	Oriental TeX crosses the Rubicon. Advanced Qur'anic Typesetting in MkIV.	talk	30	English
15:00	Hans Hagen	Arabic paragraphs	talk	15	English
15:15	Coffee			30	
15:45		Opening TeXperience 2010		10	
15:55	Jaroslav Hajtmar	The ScanCSV.lua Library: http://public.hajtmar.com/files/tex/scansv.lua/demo-scansv.lua.zip	talk	45	Czech/English
16:40	Tomáš Hála	Marking of Proof-sheets in Publishing Practice and Its Implementation in TeX System	talk	20	Czech
17:00	Roman Trušník	Typesetting Bibliography of American Literature in Czech Translation: 2000 & 2010	talk	30	English
17:30	Jan Přichystal	TeX Typesetting on Web: http://tex.mendelu.cz/en	talk	30	English
18:00	Dinner			90	
19:30	Zdeněk Wagner / Anshuman Pandey / Jaya Saraswati	Xindy Sort and Merge Rules for Indic languages: http://icebearsoft.euweb.cz/xindy-devanagari	talk	60	English
20:30	Late tea			15	
20:45		TeXperience 2010 Book Contest		30	Czech
21:15	Milan Štourač	Introducing New Traveler's Book		15	Czech
21:30	Milan Štourač	Wanderer's Notebook: Iran and Azerbaijan		90	Czech

Sunday 19.9.

8:00	Breakfast			60	
09:00	Petr Olšák	Typesetting Math: Internal Algorithms in TeX	tutorial	90	Czech
10:30	Tea			15	
10:45	Jan Štěpnička / Jan Šustek	Using TeX for Organizing International Mathematical Competition	talk	45	Czech
11:30	Mirek Olšák	CerTeXicate – Online Printing of High School Certificates	talk	15	Czech
11:45	Jan Šustek	Macros Which Handle Arithmetics with Big Numbers	talk	15	English/Czech
12:00	Jan Štěpnička	LaTeX2rtf – TeX in Building Industry	talk	15	Czech
12:15	Miloš Břejcha	Pilsen – Venue for TeXperience 2011: http://www.plzen.eu	talk	15	Czech
12:30		Viva TeXperience 2011!			
12:30	Lunch			90	

Selected Abstracts from T_EXperience

Jan Štěpnička, Jan Šustek:

Using T_EX for Organizing Vojtěch Jarník International Mathematical Competition

The University of Ostrava organizes Vojtěch Jarník International Mathematical Competition every year. Organization of the competition contains of many tasks. A large part of them is done by T_EX. On the lecture we briefly present the whole process from registration of participants to putting the results on the Internet. We describe several parts which can be useful for other T_EX users:

- during generation of the printing before the competition: simple loading of data from database, writing Unicode characters to a file or generating pseudorandom numbers,
- during procession of problems proposed for the competition: using a single source file for six different outputs or simple ignoring of macros and environments,
- during procession of results of the competition: loading results from spreadsheet or generation of diplomas.

It is likely that some of these parts are already implemented somewhere in the set of L^AT_EX packages. In this case the lecture shows an alternative approach.

Zdeněk Wagner, Anshuman Pandey and Jaya Saraswati:

Development of xindy Sort and Merge Rules for Indic languages

An index is an important part of a book or a longer document. Normally in the past the index was prepared using the MakeIndex program which offered sorting according to English and German rules. A derived program called CsIndex allowed for the preparation of indices in the Czech and Slovak languages. Sorting in other languages was difficult because the algorithm was hard coded in the C program. The situation was changed in MakeIndex 3.x which is now superseded by xindy. The sorting algorithm is now defined in tables that are present in standalone modules.

This contribution shows how xindy features can be deployed in sorting Indic languages where complex scripts are used. Since the original T_EX was unaware of Unicode, transliteration schemes were introduced in the past. X_YT_EX is quite popular mainly among new users and the transliteration systems are still in current use. Xindy is able to handle the input in the transliteration as well as the UTF-8 encoded text. The algorithm is demonstrated in the Hindi and Marathi languages. It also shows how xindy integrates with different T_EX engines.

David Březina:

Skolar – Designing a Typeface for Academic Publications

Skolar is a text serif, originally designed with scholarly and multilingual publications in mind. The typeface maintains its credibility while incorporating a subtle personal style, neither neutral nor conspicuous.

Prominent serifs and low-contrast modulation add to its robustness, and, together with a relatively large x-height, improve the typeface's readability in small sizes. This family of three weights with their respective italics and large character set is flexible enough for complex text settings and editorial work. It also becomes distinctive in bigger sizes, fitting the demands of corporate design.

There have been many practical solutions introduced in the typeface; the capitals are rather low in comparison to the ascenders. This gives the typeface even texture and more space for capital diacritical marks. The italic has a shallow angle and large counters for better readability in small print. It is easily recognized but not ostentatious, blending well with the uprights. Semibold is weighted to emphasize text blocks, where Bold is intended for word clusters. The family includes a complex set of smart arrows which can be easily keyed and infinitely combined using OpenType features. It was released with TypeTogether and it is available for web using Typekit as well.

Roman Trušník:

Typesetting *Bibliography of American Literature* in Czech Translation: 2000 & 2010

After the experience with data processing and typesetting the first part of *Bibliography of American Literature in Czech Translation* (Olomouc: Votobia 2000, 3 volumes, 1882 pages) in the late 1990s in Aldus PageMaker 5.0, new possibilities were explored before the preparation of the second part (to be published in 2010). The paper deals with the issues that had to be addressed as requirements included seamless processing of large multi-language multi-alphabet multi-font structured documents. After extensive testing (several monographs and volumes of conference proceedings, *Moravian Journal of Literature and Film*), X_YL^AT_EX was adopted as the typesetting platform for the project.

Jan Přichystal: T_EX Typesetting on Web: tex.mendelu.cz/en

This talk introduces web application T_EXonWeb. This interface helps beginners to start with typesetting system T_EX offering them easy to access and easy to use editor, T_EX compiler, code wizards and document templates. Application could also be helpful in situations when user wants to produce high-quality document but no computer with T_EX is available. Introduction to used technologies, features and future visions will also be included.

Jaroslav Hajtmar: The ScanCSV.lua Library

In computerised data processing, data stored in CSV files (Comma Separated Values) are used frequently. The presentation describes the author's library ScanCSV.lua and the method of its creation. The practical examples of its use in $\text{CON}\text{T}\text{E}\text{X}\text{T}$ MkIV will also be demonstrated. The author shows how easy and swift is to create reports, letters, forms, certificates, invitations, business cards, double-sided cards, tables, animations, etc. using external CSV databases. Users of $\text{CON}\text{T}\text{E}\text{X}\text{T}$ MkIV (but $\text{L}\text{u}\text{a}\text{L}\text{A}\text{T}\text{E}\text{X}$ and $\text{L}\text{u}\text{a}\text{T}\text{E}\text{X}$ too) can practically use data from external CSV tables in own documents through the TEX macros built by this library on-the-fly. It also means that we have this data available in an attractive, simple and natural way.

Jan Šustek: Macros That Handle Arithmetics with Big Numbers

In procedural programming languages a program calls functions with their arguments and the functions return their result. To avoid collisions, functions have their local variables. Result of a function is one of the local variables, but one can see it “nonlocally” immediately after the function call.

On the contrary, programming in TEX is based on expansions. Sequence of tokens is repeatedly expanded and when a particular token cannot be expanded, the main processor does the corresponding activity. The concept of result of a function does not make sense in TEX . If a definition, counter etc. is not local, then it is global.

We will show how it is possible, using expansions in TEX , to simulate function calls similarly as in procedural programming languages. The problem of (non)-local variables arises when one function calls another function. In this situation one can bypass the problem by a neat choice of macro names. But this is a nonnatural procedure and in the case when a function calls recursively itself it is not possible.

When one works with big numbers on computer, array is the most suitable data structure. Usual procedural programming languages know arrays. TEX primarily does not know arrays. We show one of possible implementations of arrays in TEX . This implementation, however, is not good for sending the values between function. Hence we show another data structure and functions for conversion between different data structures. It is senseless to write that decadic expansion is not suitable.

As a demonstration of the mentioned function calls we implemented functions from number theory. The following short programs allows us to decide whether c is composite.

$$\begin{aligned} c &= (2^{107} - 1)(2^{127} - 1) = \\ &= 27606985387162255149739023449107931668458716142620601169954803000803329 \end{aligned}$$

```
\SET\a\POWER(2)(107)
\SET\a\SUBTRACT[\a](1)
\SET\b\POWER(2)(127)
\SET\b\SUBTRACT[\b](1)
\SET\c\ULTIPLY[\a][\b]
\ArrayToPrint\c\c
\SHOW c
\SET\r\ISCOMPOSITE(\c)
\SHOW r
\bye
```

Function `\ISCOMPOSITE` performs Fermat test calling function `\FERMAT`, which computes power in modular arithmetic calling function `\POWERMOD`, which multiplies its arguments calling function `\SQUAREMOD`, which, when looking for remainder, uses division calling function `\DIVIDE`, which uses subtraction calling function `\SUBTRACT`.

Abstracts without Papers

Taco Hoekwater: Metapost Development

Metapost 2.000 is planned for release in the summer of 2010. This presentation is a short report on the current status of the development. Metapost version 1.500 will (hopefully) be ready just in time for BachoT_EX 2010, and in that release all memory arrays will have been replaced by dynamic memory allocation. A followup in the evening program highlights some of the ugly details.

Taco Hoekwater: Lua for Beginners

A tutorial that explains the basics of Lua programming and demonstrates some ways it can be used within documents.

Taco Hoekwater: Lua for Font Lovers

A tutorial that explains how the LuaT_EX fontloader library works, and shows various font tricks you can do by using Lua code.

Taco Hoekwater:

Escrito – A PostScript-compatible Interpreter in Lua

Escrito is the name of an interpreter for the PostScript language written in pure Lua code. Its default output device generates Portable Document Format (PDF) operators, making it ideally suitable for handling PostScript code within LuaT_EX.

Using a little bit of Lua glue code, it will become possible to do the following things in LuaT_EX that were previously only doable via complex workarounds:

- * Include EPS images without the need for an external conversion process.
- * Support the PSTricks macro package.
- * Run Kees van der Laan's PostScript programs in-line.

Taco Hoekwater: tcontrib.metatex.org

TL Contrib is a repository for packages and package updates that for various reasons cannot be in T_EX Live itself. Think of packages that not quite free software according to the debian guidelines (for example packages that prohibit changes or come without sources), packages that prohibit commercial resale (the T_EX Live DVDs are sold by Lehmanns), and updates for binary packages that actually are in T_EX Live. The TL Contrib repository can be used with tlmgr to install such packages, and of course it is possible for authors to register their packages in it.

Mojca Miklavc: ConT_EXt Minimals, “Server Edition”

The idea is to release clean code by the time of conference that would enable anyone to set up his own mirror for ConT_EXt distribution and thus increase its bus factor in case that Mojca goes to long vacations or becomes too inspired by her research work.

Mojca Miklavc and Hans Hagen: The Database Module, MkIV Version

As Hans promised Mojca to make a MkIV variant of the database module that she loves so dearly, an actually did it, this variant will be presented at the conference by Mojca and Hans.

David Březina: General Issues in Multi-script Typography

An overview of the most common typographic issues in publications which involve two or more scripts with a horizontal reading direction (e.g. Latin, Indic, Arabic, Hebrew, etc.). The talk aims to explain the problems systematically from a partially-objective point of view and show some of the possible solutions.

Luigi Scarso: Modules_mkiv

Modules_mkiv is an attempt to document context source code in an automatic fashion, and also a test for LuaT_EX and ConT_EXt MkIV. All PDFs are made with ConT_EXt MkIV and LuaT_EX, see

<http://foundry.supelec.fr/gf/project/modules/>.

Hans Hagen: ConT_EXt Lua Documents

For a while now we have the ConT_EXt user interface accessible at the Lua end. Although future releases will have an additional (and different) library, the current one is especially handy for users who are already familiar with ConT_EXt. The CLD interface is rather simple and can be quite powerful but it also has some limitations. All of this will be discussed.

Hans Hagen: Font Goodies

The ability to have control over matters has always been one of T_EX’s virtues. Operating beyond what fonts provide out of the box is an example of this. Fonts can have features that users turn on or off, but how to know what to apply when? And how about additional features? Do we want detailed control at the T_EX end or can it be done more conveniently in Lua? Here I will introduce ‘font goodies’ as a way out. Don’t hesitate to share your wish list during this presentation.

Hans Hagen: Why Structure Matters

There have always been lots of structure related commands in `CONTEXT`. In `MkIV` much of the low level code has been reimplemented with future extensibility in mind. This presentation will give an update of the status and a preview of the future.

Hans Hagen: How LuaTeX And ConTeXt Proceed

As the development of `LuaTeX` and `CONTEXT` sort of go hand in hand there is an ongoing change in the code that makes up `CONTEXT`. Some of the changes and extensions are just proof of concept, others can will stay and can be used freely. In this presentation I will discuss the things that were done recently.

Hans Hagen: Whatever You Want to Know

As usual there can and will be sessions where you can ask whatever you like about `CONTEXT`, the (probably mostly unknown) tools that come with it, how it is used or can be used, etc. Feel free to submit your questions in advance.

Hans Hagen: Requirements for Documentation

`CONTEXT` evolved out of our own usage: educational documents and general documentation. This is why it is organized as it is. I will discuss a couple of characteristics and how they can help you to separate content from rendering issues. I will show some sources and styles of manuals that relate to context.

Hans Hagen: My Slowly Growing Test Suite

It has become good practice that users post small examples on the mailing list when they run into problems or provide solutions. I also make small test files myself and for a while I've been collecting them. They are available to users and can also serve as showcases. In due time Luigi will add the output to the module collection that he maintains. I will give some demonstrations and we can discuss how to make more examples and how to organize them.

Hans Hagen: XML Processing News

Part of the `MkIV` XML processing code has been rewritten. I will discuss and demonstrate a few of the new (and changed) features. (Maybe I can/will use the experimental `MkIV` `BIBTeX` code as an example.)

Hans Hagen: A ConTeXt Scoop

Normally new features are put into the kernel when they get stable. However, I decided to keep the latest trick out till the conference so that those who attend get the first view of it. Let's see how long they can keep the secret.

Patrick Gundlach:**Lua \TeX without \TeX – or: the Hidden Beauty of \TeX**

With Lua \TeX you can typeset text without having to deal with \TeX 's input language. This is great, because you don't need to take care of catcodes anymore, you don't have to `\relax` after dealing with complex macros, expansion is not an issue anymore. In this talk I will present how to typeset text just by using Lua(\TeX)-functions and I will show some examples, such as the `\boxit` macro from the \TeX book and how this could be done in Lua. This presentation does not deal with $\text{CON}\TeX\text{T}$ at all.

Patrick Gundlach: contextgarden.net

This is a question and answer session on the current state of `contextgarden.net`. This can be done in the evening when we have some tea or coffee.

Alan Braslau: Drawing Chemical Structures Using ppch \TeX

Chemical formulas and chemical structures can be included in a \LaTeX or a $\text{CON}\TeX\text{T}$ document easily using the ppch \TeX macros. I present a simple introduction to their use.

This package has been completely re-written in MkIV (not by me!) and is now included in the core macros. The aim of this presentation is to stimulate discussion as some further development could be useful.

Alan Braslau: Plotting Data with Metafun/Metapost

Data can be graphically presented using the `METAPOST/graph` macros. I present here a simple introduction to their use in order to stimulate discussion on the interest of these macros and to explore alternative solutions including the possibilities to process numerical data through use of Lua \TeX .

Alan Braslau: Drawing Diagrams Using the Chart Module

$\text{CON}\TeX\text{T}$ provides a `charts` module to create flow charts. This module could be extended to abstract positioning, somewhat akin to the drawing of tree diagrams, very easily performed using `TikZ`. Other examples of diagrams of arbitrary connected cells such as pneumatic or vacuum systems will be presented.

Willi Egger: Arranging Pages for Printing

Although e-readers are coming up it is still so, that typeset information very often will have to be printed on paper. In order to make a final product one

needs to arrange the contents in such a way, that after folding the product makes a professional impression. `CONTEX`T does not only do a great job by typesetting, it offers also solutions to create a final product which can directly be fed to a printer in the print house. In this session I will give an overview on the currently available imposition schemes with special emphasis on flyers in different presentations.

Wolfgang Schuster: The Letter Module

Where i got inspiration for the implementation (`m-letter.tex` and `letterstyle.tex`)

- * layer system of the module (from internals to user interface),
- * the user interface,
- * workshop how to create your own style.

Wolfgang Schuster: Module Documentation

- * Methods to document a module (in the code and as a separate document).
- * How create your own command definitions (what you can see in `setup-en.pdf/cont-en.xml`).

Arthur Reutenauer: The Ever-regenerating Hydra – Hyphenation Patterns in Unicode, and beyond

The `hyph-utf8` project was launched in the spring of 2008, as a means of rationalizing the situation of hyphenation patterns in `TEX` Live. Its goal at the time was to convert all the patterns to UTF-8 so that `XYTEX` and `LuaTEX` could use them directly, and to make them also available to non-Unicode-aware `TEX` engines by converting them on the fly, in order to keep backward compatibility. At that point, this effort was only useful to `LATEX`, because `CONTEX`T had been using its own copy of the patterns, but it ultimately benefited from `hyph-utf8` as well, as it could use it as the sole source for hyphenation patterns.

Since we started, the project's scope has far outgrown its original purpose: first, it was integrated in `MiKTEX` shortly after we began (while our target at the time was only `TEX` Live); and we also had the opportunity to add patterns for languages that we never had before, coming from different sources. In particular, several languages we added were, at best, extremely awkward to support in a non-Unicode environment, and we thus decided to support them only as UTF-8, in `XYTEX` and `LuaTEX`, without attempting to convert them to some 8-bit encoding. This approach will hopefully help the `TEX` community in supporting these languages better.

We also had the chance to work with other projects that use the same hyphenation algorithm as `TEX`: `OpenOffice`, for starters, whose relationship to `TEX`

is obvious, is the source for several of the “new” languages. There are also several implementations of the algorithm for Web browsers, with which we could exchange material: their primary source for patterns are always T_EX distributions, but they also received files from individual contributors; we thus added a couple more languages thanks to them. Another program that uses our patterns is Apache FOP (Formatting Object Processor), a document formatter driven by XSL-FO.

It is our hope that hyph-utf8 can become a central source of material and information on word hyphenation for all projects that may need them.

Karel Horák: Do It Better...

Collection of some mostly practical problems with solutions (often borrowed from other people’s ideas) which all have one thing in common: get efficiently any typesetting as good as possible.

Idris Samawi Hamid: Towards the First ConT_EXt Book

Given the breadth of CON_T_EX_T, and the growth of the CON_T_EX_T community, the need for a polished book that introduces its scope and power has become ever more pressing. In this talk we intend to introduce an outline of this first major CON_T_EX_T book project, and to get input and suggestions from the participants to help make it successful.

Idris Samawi Hamid: Oriental T_EX Crosses the Rubicon. Advanced Qur’ānic Typesetting in MkIV

After years of research, the Oriental T_EX Project can proudly announce that it is closing in on the holy grail of paragraph-based Arabic typography. We illustrate this by demonstrating the typesetting of the Qur’ān in LuaT_EX and CON_T_EX_T MkIV. We also discuss some details of OpenType typesetting control in MkIV as well.

Mobile T_EX: Porting T_EX to the iPad

T_EX mobilně: migrace T_EXu na iPad

ARTHUR REUTENAUER

Abstract: The paper presents the achievement of Richard Koch, amongst others author of T_EXShop and MacT_EX developer, who has successfully compiled and used T_EX on Apple's iPad.

Key words: T_EX-8, C_ON_TE_XT, iPad.

Abstrakt: Článek nám představí úspěch Richarda Kocha, mimo jiné autora programu T_EXShop a distribuce MacT_EX, kterému se podařilo zkompileovat a použít T_EX na iPadu od firmy Apple.

Klíčová slova: T_EX-8, C_ON_TE_XT, iPad.

*arthur (dot) reutenauer (at) normalesup (dot) org
Allée du Torrent
Zone Tokoro
05000 GAP
France*

Introduction

Over the 2⁵ years of its existence, T_EX has been ran on many platforms and has always been noted for its portability, so it's not surprising that when new devices appear, T_EX would soon be ported there, too.

Today a new kind of device is in wide use that links computers to telephones and that represents a new challenge on the path of T_EX because of reasons both technical and ergonomic, the so-called “smartphones”. But it may sound insane to want to use T_EX on a Blackberry or a Nokia N97—although word on the street is that Jonathan Kew, author of X_YT_EX and T_EXworks, ported T_EX to the iPhone last year—so that's not exactly what I will talk about here.

I will present the achievement of Richard Koch, amongst others author of T_EXShop and MacT_EX developer, who has successfully compiled and used T_EX on Apple's iPad.

The latter does of course not qualify as a smartphone per se, but it shares a lot of features with them, above all the aim at mobility, while having the advantage of giving the user an experience closer to that of an actual computer. The iPad port of T_EX, called T_EX-8, should therefore give a good idea of what “mobile T_EX” could be. It might even be that T_EX-8 could be copied to the other Apple mobile devices with only minor changes (who wouldn't want to run T_EX on an iPod touch?), but I will stick to describing the experiments I made, thanks to Richard, on the iPad, and hint that they would probably also apply to the other iThings.

There are a number of issues arising from this task, that indeed could be qualified as Herculean. The way programs work on the iPad is that they're made into “applications” from which you control everything. It is the single entry point to the program, and in fact the only way we can interact with the operating system, because we can't run two applications at the same time. This of course doesn't mean that we can't program many things into an application, but it gives a very different touch: for T_EX to be made into an iPad application, we have to embed not only the T_EX program itself, the format file and the whole distribution into it, but also the editor itself! If we used a different application to edit the source file, T_EX-8 couldn't access it because of Apple's very restrictive policy.

Hence, the interesting issue is that, while all the pieces we need are obviously already available, and the story of writing an iPad application for T_EX mostly is the description of how we put the pieces together, it also consists not in small part in crucial design choices aiming at crafting a reasonable user interface.

A word of warning before we proceed to the description proper: I wanted to introduce this project today but it is yet very much a work in progress, and I cannot present anything else than a snapshot of the current development stage. I thus ask the reader to take the following pages with a little bit of salt. All the advertised features, or lack thereof, I review here, will, no doubt, be greatly improved in the future.

So here goes.

A user-friendly interface

A rich text editing environment

The natural entry point to $\text{T}_\text{E}\text{X}$ on the iPad seems to be the source code, and $\text{T}_\text{E}\text{X}$ -8 thus opens on the editor window, that displays by default a document stored in the application about Sylow theorems in group theory. The whole window is a text area, and the iPad virtual keyboard pops up, so that we can type. Typing on the iPad is “not for sissies”, in Richard’s words, and typing $\text{T}_\text{E}\text{X}$ code is rendered even more awkward by the fact that many of $\text{T}_\text{E}\text{X}$ ’s special characters that are ubiquitous (`‘’‘’`, amongst several others) are not present on the default keyboard, but demand instead, in order to be typed, that one switch keyboard *twice* (using two different toggle keys), which would make the typing of any serious $\text{T}_\text{E}\text{X}$ document extremely painful. In order to remedy that, Richard devised an additional row on top of the standard keyboard, that pops up and disappears together with it. Though I personally regret that the simulated keys don’t make the nice keyboard-like sound when tapped, they are immensely helpful and make for a reasonable tradeoff if one needs to type actual input on the iPad.

Another standard iPad feature that could come in handy at that point would be auto-correction of the typed input. It behaves like some kind of aggressive spell-checker, automatically correcting any words the user types, unless the latter directs otherwise. Having personally experienced how this behaviour can have very disruptive effects in some places (when typing URLs, for example), I don’t especially recommend to turn it on by default, but it can be useful if one is typing lengthy text in a natural language, and could have spared me to have written “The whole ndow ia a text qrea, and the ipad keybaoadr popz up, so thqt we cqntpe.” when I first typed the above paragraph.

Apart from that feature, the editor is very basic since it borrows from the standard TextEdit application for typing plain text, and has therefore none of the capabilities one may expect from a development environment for $\text{T}_\text{E}\text{X}$, apart from a “typeset” button that has the obvious effect. Encoding support is also very poor.

It should also be noted that for the moment, the $\text{T}_\text{E}\text{X}$ run freezes everything in the editor and doesn’t stop until it completes its task: one can neither use the editor, nor interrupt $\text{T}_\text{E}\text{X}$ as it runs.

A convenient file browser

It is yet quite cumbersome to upload files on the iPad. As I have not tried it personally—or rather I did try, but nothing worked—I simply copy the notes by Richard:

My intention is that there will be three ways to proceed:

- a. *Load and unload in iTunes, connecting to a regular Mac.*
- b. *Mail source and output into the iPad and back out.*
- c. **Other** *suitable programs on the iPad can send source to $\text{T}_\text{E}\text{X}$ -8, and $\text{T}_\text{E}\text{X}$ -8 can send source and output to them. Then these programs can communicate with the outside world.*

But c) isn't working at all, and b) is only working in the sense that you can mail both source and PDF out of \TeX -8.

The iTunes method seems to be currently limited by Apple. Here's how it works. When you connect the iPad to iTunes, you'll see a list of programs which can communicate, and \TeX -8 will be listed. You will see a list of folders, one for each program. For instance, one folder will be labeled "Mordell". You cannot look inside these folders in iTunes, but you can drag one of the iTunes window to your desktop or elsewhere. If you drag, say Mordell, then you'll get a folder containing the source, the PDF output, the log and sync files, and all the illustrations. So this method is clumsy but works to get stuff out of the iPad.

Unfortunately, Apple's software doesn't allow you to drag folders back in. You can only drag individual files in. I suppose the way to handle that is to zip up a folder and then drag the zip in and have \TeX -8 unzip it to a folder the next time it runs. But for now, here's what you can do:

Drag your source and all supplemental files (i.e., illustrations) to \TeX -8 in iTunes. Make sure you only drag **one** .tex file (no input tex files allowed). The next time \TeX -8 starts, it will look for a .tex file. If one and only one exists, it will create a folder with that name and put all the other files in that folder.

That's the only current method to get illustrations into the machine.

A feature-full previewer

The preview window allows us to see the PDF file produced by \TeX (for the moment, we use PDF as the only output format). We can slide back and forth through the pages, using a standard iPad feature. There is also a button to go to a specific page number, and two other buttons, to jump to the first and the last page. And that's all.

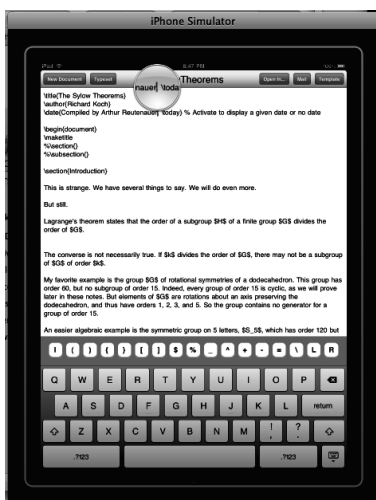


Figure 1 Editing a \TeX file on the iPad

No, really. That's all.

A this point of development, there is no search feature and also no magnification possible of the page. And, as has already been hinted, the typesetting process gives absolutely no feedback to the user: \TeX doesn't display the output as it does on other systems, and the log file, though stored on disk as usual, is not accessible at all. What's more, if compilation fails but some PDF file was already produced by earlier runs, the user is presented with that file, which is a rather confusing behaviour.

The diligent reader may wonder what " \TeX " we really used in all of the above, so let the audience be reassured: we can *of course* use Con \TeX t with \TeX -8, just as \LaTeX and plain \TeX ; for the moment I only experimented with Mark II, though, as I have only been able to use pdf \TeX as the underlying engine.

A well-designed programming interface

Now to some technical issues: the main problem one is faced when making \TeX into a iPad application is that you simply can't run \TeX as a separate process from the application. Apple policy forbids it. You need to make a library out of the \TeX program, and to call the `main` function (the entry point to any program written in C) from the application. And \TeX is not thought out for that use, which makes that task awkward, even if of course possible. A funny issue is that, for example, in the sources for pdf \TeX which Richard took, there are two functions that are called `main` (in `texk/web2c/lib/main.c` and `texk/web2c/lib/texmfmp.c` in the source hierarchy, respectively); but of course only one is the real entry point to the \TeX program. Another much more serious problem is memory management. Because traditionally each \TeX run has been autonomous, \TeX 's memory doesn't need to be managed so meticulously and one can rely on the operating system to clean up \TeX 's memory upon exit, because \TeX calls the C library function that is in fact called `exit` and that takes care of that. When \TeX is used as a library, though, calling `exit` shuts down the entire program, therefore killing the editing window, and returning to the iPad "Home Screen", which is a bit ridiculous; and while the solution to this particular problem is obvious and immediate to implement (just remove the call to `exit`), the underlying problem remains: \TeX 's memory isn't cleaned up and the system loses track of it after the processing, leading to *memory leaks*. These leaks are in fact so important that the limited iPad memory (256 megabytes), while vastly sufficient for any single run of \TeX , can't handle the next run and, as of today, *you can't yet run \TeX twice in a row*. You need to quit the application and relaunch it. Thankfully, the application reopens in the exact same state as it was when we left it. This is of course far from optimal, and unlike most of the problems outlined above, it will take work to be solved; but we shouldn't despair.

Other than that, the typesetting speed is reasonable (for the first and only run), which is not that surprising: though limited compared to today's computers, the iPad resources are still immensely more powerful than what was available thirty years ago, when \TeX was first developed.

Finally, as has already been said, we used pdf \TeX ; I have managed to compile Lua \TeX as a library as to use for compiling simple documents, but, aside from having one more `main` function (in `texk/web2c/luatexdir/luatex.c`, as it is), the memory management problems are even

worse than with pdfT_EX, and when I tried it, the application systematically crashed towards the end of the run.

Conclusion

This is thus the point where we are now. As a conclusion I would like to quote Richard's words:

"[...] in the iPad world, I don't know the right approach. This machine seems like a new category to me—one that will take several years of experiments before we know what works best. It is more a minimalist machine, and I suspect the best programs will be those that throw away features we have come to depend upon to concentrate on a few new paradigms. But I don't know what they will be."

Having been lucky enough to test the application and to use it to typeset a few documents of my own, I fully subscribe to this opinion, of course: the user interface is lacking in many respect at this early stage of development stage, but it will take (programmer) time and (user) experience to improve it and make it really usable.

One thing we certainly shouldn't do, for example, is to port indiscriminately all the capabilities of T_EX editors and PDF previewer directly to the iPad application, that would certainly not give a very enjoyable experience.

As a simple example, let us consider the problem of error reporting. While dealing with the problem of a faulty T_EX run is straightforward (we just display an error message to the user if the PDF file is not newer than the T_EX source), it is not immediately obvious—at least to me—how to display the log file to the user: it should of course be available in some way to people wanting to know what exactly went wrong, but do we need to display the full output of the T_EX run, like we have when we run T_EX in a terminal window, or from a specialized editor like T_EXMaker or T_EXworks? I'm not sure it is really useful, nor even desirable, on such a minimalist device as the iPad (not to mention actual smartphones, where it would probably be unreadable). Thus, for the moment, T_EX runs in "non-stop mode", as if one had typed 's' during an interactive run, and returns even if it produces no PDF (as opposed to waiting for the user's input to complete its task).

And when it comes to the future, I like to think that experimenting with T_EX on unusual platforms is a chance to think about new ways to use T_EX, and that ConT_EXt, that has already been providing complete chaintools in T_EX processing for decades, is certainly well prepared for such a challenge!

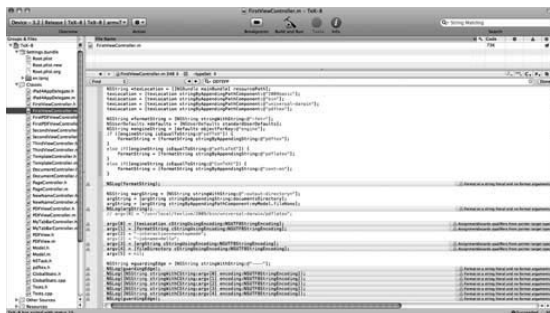


Figure 2 Xcode sources for T_EX-8



Figure 3 The \TeX -8 application in its natural habitat

Playing with Flash in ConT_EXt-mkiv

Animace ve Flashi pomocí ConT_EXt-mkiv

LUIGI SCARSO

Abstract: Starting from release 9 AdobeReader, the reference PDF viewer from Adobe, has a Flash player embedded. The recent addition to CTAN of flashmovie package by Timo Hartmann prompted me to investigate the feasibility of an integration between ConT_EXt-mkiv and swf figures. All tests were performed under Linux Ubuntu 8.04 with AdobeReader 9.3.3 installed, but I suppose they also work under Windows or Mac operating systems.

Keywords: Flash, Flash animation, ConT_EXt, Mark IV, Lua

Abstrakt: Článek představuje jednu z možností, jak vložit animace vytvořené ve Flashi do PDF. Používá na to ConT_EXt-mkiv, který využívá programovacího jazyka Lua.

Klíčová slova: Flash, animace ve Flashi, ConT_EXt, Mark IV, jazyk Lua

Introduction

In the Issue 2010, Number 1 of th PracT_EX journal I have published a first tentative to support SWF files in ConT_EXt-mkiv. It was a literal translation of the `flashmovie.sty` stylesheet [2] and the result was an unusual mix of pdfL_AT_EX and ConT_EXt-mkiv code, but the main reason was to gain a good knowledge of the specifications and to test some applications. Just before the article was published Hans translated the stylesheet into the ConT_EXt-mkiv lingo, so ConT_EXt users can already use the swf files as figures: what I suggest here is a *all-or-nothing* way to implement the requirements of specification and also show some applications.

Implementation

The first step to support SWF files as external figures in ConT_EXt-mkiv is to register the swf interface into the `grph-inc.mkiv` file with `\registercctxluafile:`

```
grph-inc.mkiv:
\registerctxluafile{grph-swf}{1.001} % this will change
```

Inside `grph-swf.lua` the function `figures.checkers.swf(data)` inserts the annotation object that identifies the swf figure using the good old `\pdfannotation` macro instead of a Lua function `node.write(pdfannotation(width,-height,0,annot()))` (the code is commented, as one can see), but it's one of the fews still present:

```
grph-swf.lua:
local format = string.format
local texsprnt = tex.sprnt
local ctxcatcodes = tex.ctxcatcodes
local pdfannotation = nodes.pdfannotation
function figures.checkers.swf(data)
  local dr, du, ds = data.request, data.used, data.status
  local width = (dr.width or figures.defaultwidth):todimen()
  local height = (dr.height or figures.defaultheight):todimen()
  local foundname = du.fullname
  local controls = dr.controls or nil
  local display = dr.display or nil
  dr.width, dr.height = width, height
  du.width, du.height, du.foundname = width, height, foundname
  texsprnt(ctxcatcodes,format(
    "\\ startfoundexternalfigure{%ssp}{%ssp}",width,height))
  local annot, preview, ref = backends.pdf.helpers.insertswf {
    foundname = foundname,
    width      = width,
    height     = height,
    -- factor   = number.dimenfactors.bp,
    display    = display,
    controls   = controls,
    -- label    = dr.label,
  }
  -- node.write(pdfannotation(width,-height,0,annot()))
  texsprnt(ctxcatcodes,format("\\ pdfannot width %ssp height %ssp {%s}",
    width,height,annot()))
-- brrrr
  texsprnt(ctxcatcodes,"\\ stopfoundexternalfigure")
  return data
end
figures.includers.swf = figures.includers.nongeneric
figures.registersuffix("swf","swf")
```

Actually the code before is modification of mine, where I've simply un-commented the `display` and `controls` variables because I need them later. The complete specifications consist of the PDF Reference sixth edition book and Adobe® Supplement to the ISO 32000 BaseVersion: 1.7 ExtensionLevel: 3 both available

from [1]. The chapter 9.6 Rich Media of the Supplement describes the additional entries of the RichMedia annotation dictionary, and it's the guide to what follow; carefully reading of the RichMedia chapter and the code below reveals that there is almost an one-to-one map between the specifications and the implementation that is done by the Lua tables:

```

local format = string.format
local pdfconstant = lpdf.constant
local pdfboolean = lpdf.boolean
local pdfstring = lpdf.string
local pdfunicode = lpdf.unicode
local pdfdictionary = lpdf.dictionary
local pdfarray = lpdf.array
local pdfnull = lpdf.null
local pdfreference = lpdf.reference
function backends.pdf.helpers.insertswf(spec)
    local width, height, filename = spec.width, spec.height, spec.filename
    local controls = spec.controls or nil
    local display = spec.display or nil
    if controls = 'no' then
        if (parametersets[controls].replace_helper == true) and
            (type(parametersets[controls].private_helper) == "function") then
            local annotation
            annotation = parametersets[controls].private_helper(spec)
            return annotation,nil,nil
        end
    end
end
local eref = backends.codeinjections.embedfile(filename)
local configuration = pdfdictionary {
    Type = pdfconstant("RichMediaConfiguration"),
    Subtype = pdfconstant("Flash"),
    Instances = pdfarray {
        pdfdictionary {
            Type = pdfconstant("RichMediaInstance"),
            Subtype = pdfconstant("Flash"),
            Params = pdfdictionary {
                Type = pdfconstant("RichMediaParams"),
                -- FlashVars =
                Binding = pdfconstant("Foreground")
            },
            Asset = eref
        },
    },
}
local configuration_ref = pdfreference(pdf.immediateobj(tostring(configuration)))
local content = pdfdictionary {
    Type = pdfconstant("RichMediaContent"),
    Assets = pdfdictionary {
        Names = pdfarray {
            pdfstring(filename),

```

```

        eref,
    }
},
Contents = pdfarray { configuration_ref },
}
local content_ref = pdfreference(pdf.immediateobj(tostring(content)))
local settings = pdfdictionary {
    Type = pdfconstant("RichMediaSettings") ,
    Activation = pdfdictionary {
        Type = pdfconstant("RichMediaActivation"),
        Condition = pdfconstant("PO"),
        Animation = pdfdictionary {
            Subtype = pdfconstant("Linear"),
            Playcount = 1,
            Speed = 1,
        },
    },
    Configuration = configuration_ref,
    Presentation = pdfdictionary {
        PassContextClick = true,
        Style = pdfconstant("Embedded"),
        Toolbar = false,
        NavigationPane = false,
        Transparent = true,
        Window = pdfdictionary {
            Type = pdfconstant("RichMediaWindow"),
            Width = pdfdictionary {
                Default = 100,
                Min = 100,
                Max = 100,
            },
            Height = pdfdictionary {
                Default = 100,
                Min = 100,
                Max = 100,
            },
            Position = pdfdictionary {
                Type = pdfconstant("RichMediaPosition"),
                HAlign = pdfconstant("Near"),
                VAlign = pdfconstant("Near"),
                HOffset = 0,
                VOffset = 0,
            }
        }
    },
},
Deactivation = pdfdictionary {
    Type = pdfconstant("RichMediaDeactivation"),
    Condition = pdfconstant("XD"),
},
}

```



```

local settings_ref = pdfreference(pdf.immediateobj(tostring(settings)))
local annotation = pdfdictionary {
  Subtype          = pdfconstant("RichMedia"),
  RichMediaSettings = settings_ref,
  RichMediaContent = content_ref,
}
return annotation, nil, nil
end

```

RichMedia annotations have a huge set of options and the more convenient way to manage them is by a Lua table: ConTEXT-mkiv has an experimental mechanism that uses the global table `parametersets` to store and retrieve the values. What follow is not the canonical syntax `\startluaparameterset [<namespace>] ..\stopluaparameterset` but a Lua version that is essentially the same:

```

\startluacode parametersets["swf:Main:controls:1"] = {
  replace_helper = true,
  private_helper = document.lscarlo.insertswf
\stopluacode
\externalfigure[Main.swf][width=320px,height=180px,controls=swf:Main:controls:1]

```

The idea is clear: there is only one option `controls` instead of many keys/values and this option “points” to a dictionary of keys/values. ConTEXT-mkiv has also another option `display`, because the idea is a clear separation between presentation and control, but I’ve not used it in my implementation. The standard support for `swf` figures doesn’t manage the option `controls`, so I have added my own code:

```

local controls = spec.controls or nil
local display = spec.display or nil
if controls = 'no' then
  if (parametersets[controls].replace_helper == true) and
    (type(parametersets[controls].private_helper) == "function") then
    local annotation
    annotation = parametersets[controls].private_helper(spec)
    return annotation,nil,nil
  end
end
end

```

Now it’s time to explain what I mean with “all-or-nothing”. If you want just use a `swf` figure just do *nothing*, i.e. `\externalfigure[Main.swf]` is suffice. But if you need to specify some options then you must pass them to `externalfigure` together with *all* the defaults ones, and a way to pass different options is just replace the Lua function `backends.pdf.helpers.insertswf(spec)` with a private implementation `document.lscarlo.insertswf` by storing its reference into the `swf:Main:controls:1` table. This is the meaning of

```

\startluacode
parametersets["swf:Main:controls:1"] = {
  replace_helper = true,
  private_helper = document.lscarlo.insertswf
}
\stopluacode

```

replace_helper = true is hence a signal to backends.pdf.helpers.insertswf (spec) to replace itself with the private implementation document.lscarlo.insertswf (spec). A trivial implementation of document.lscarlo.insertswf (spec) is almost a copy of the standard backends.pdf.helpers.insertswf (spec):

```

\startluacode
document.lscarlo = document.lscarlo or {}
function document.lscarlo.insertswf(spec)
  local format = string.format
  local pdfconstant = lpdf.constant
  local pdfboolean = lpdf.boolean
  local pdfstring = lpdf.string
  local pdfunicode = lpdf.unicode
  local pdfdictionary = lpdf.dictionary
  local pdfarray = lpdf.array
  local pdfnull = lpdf.null
  local pdfreference = lpdf.reference
  local width, height, filename = spec.width, spec.height, spec.foundname
  local controls = spec.controls or nil
  local display = spec.display or nil
  local eref = backends.codeinjections.embedfile(filename)
  local configuration = pdfdictionary {
    Type = pdfconstant("RichMediaConfiguration"),
    Subtype = pdfconstant("Flash"),
    Instances = pdfarray {
      pdfdictionary {
        Type = pdfconstant("RichMediaInstance"),
        Subtype = pdfconstant("Flash"),
        Params = pdfdictionary {
          Type = pdfconstant("RichMediaParams"),
          Binding = pdfconstant("Foreground")
        },
      },
      Asset = eref
    },
  },
}
local configuration_ref = pdfreference(pdf.immediateobj(tostring(configuration)))
local content = pdfdictionary {
  Type = pdfconstant("RichMediaContent"),
  Assets = pdfdictionary {
    Names = pdfarray {
      pdfstring(filename),

```

```

        eref,
    }
},
Contents = pdfarray { configuration_ref },
}
local content_ref = pdfreference(pdf.immediateobj(tostring(content)))
local settings = pdfdictionary {
    Type = pdfconstant("RichMediaSettings"),
    Activation = pdfdictionary {
        Type = pdfconstant("RichMediaActivation"),
        Condition = pdfconstant("PO"),
        Animation = pdfdictionary {
            Subtype = pdfconstant("Linear"),
            Playcount = 1,
            Speed = 1,
        },
    Configuration = configuration_ref,
    Presentation = pdfdictionary {
        PassContextClick = true,
        Style = pdfconstant("Embedded"),
        Toolbar = false,
        NavigationPane = false,
        Transparent = true,
        Window = pdfdictionary {
            Type = pdfconstant("RichMediaWindow"),
            Width = pdfdictionary {
                Default = 100,
                Min = 100,
                Max = 100,
            },
            Height = pdfdictionary {
                Default = 100,
                Min = 100,
                Max = 100,
            },
            Position = pdfdictionary {
                Type = pdfconstant("RichMediaPosition"),
                HAlign = pdfconstant("Near"),
                VAlign = pdfconstant("Near"),
                HOffset = 0,
                VOffset = 0,
            }
        }
    },
},
Deactivation = pdfdictionary {
    Type = pdfconstant("RichMediaDeactivation"),
    Condition = pdfconstant("XD"),
},
}
local settings_ref = pdfreference(pdf.immediateobj(tostring(settings)))

```

```

local annotation = pdfdictionary {
  Subtype          = pdfconstant("RichMedia"),
  RichMediaSettings = settings_ref,
  RichMediaContent = content_ref,
}
return annotation, nil, nil
end
\stopluacode

```

The rationale behind this implementation is that most of the time the user wants to specify only some keys/values, but sometimes a bit of programming is required, as for example to calculate the indirect reference of an object. Needless to say that Lua is almost perfect for this, so it seemed to me a natural solution to delegate the user to write the appropriate function (in this way he *must know* the options and their meaning) and let ConTEXt-mkiv replace the standard implementation with the user's one.

Application

As simple application, I've considered the programs `as3compile` and `swfc` from the `swftools` suite [4]. The goal is to achieve something similar to `METAPOST`: typeset the code and straight insert the result into the pdf, where in this case the code is `ActionScript3` code that is compiled into a `swf` figure with the `as3compile` compiler, an external program. The implementation is also simple: the `ActionScript` code is enclosed between a couple of `start/stopSWFtoolsAScode` macros (with some options as the name of the script and the path of the compiler) that are in turn almost a verbatim copies of `start/stopluacode` macros:

```

\long\def\dostartSWFtoolsAScode[#1]
  {\getparameters[as.][ name={out-as},preamble=preamble,compiler=as3compile,#1]%
  \begingroup
  \obeylualines %% yes, lua
  \obeyluatokens %% yes, lua
  \dodostartSWFtoolsAScode}
\long\def\dodostartSWFtoolsAScode#1\stopSWFtoolsAScode
  {\normalexpanded{\endgroup\noexpand\dododostartSWFtoolsAScode[#1]}}%
\long\def\dododostartSWFtoolsAScode[#1]{
\startluacode
  local preamble = ''
  local outfile = tostring("\csname as.name\endcsname") .. ".as"
  local swffile = tostring("\csname as.name\endcsname") .. ".swf"
  local ascompiler = tostring("\csname as.compiler\endcsname")
  local asscript_body = [=[#1]=]
  print('')
  local asscript = preamble .. asscript_body
  io.savedata(outfile,asscript)

```

```

as_execute = string.format("\%s \%s -o \%s ",ascompiler,outfile,swffile)
os.execute(as_execute)
\stopluacode%
}
\unexpanded\ef\startSWFtoolsAScode{\ostartSWFtoolsAScode} % lua catcodes
\startSWFtoolsAScode[name=smile,
compiler={/opt/luatex/minimals-2010-brejlov/tex/texmf-project/bin/as3compile}]
package
{
import flash.display.MovieClip
public class Main extends MovieClip
{
function Main()
{
this.graphics.beginFill(0xcccc00)
this.graphics.drawCircle(200,200,200)
this.graphics.endFill()
this.graphics.beginFill(0x000000)
this.graphics.drawCircle(140,150,50)
this.graphics.drawCircle(260,150,50)
this.graphics.drawRoundRect(140,270,120,10,20);
this.graphics.endFill()
}
}
}
\stopSWFtoolsAScode
\externalfigure[smile.swf] [width=100px,height=100px]

```



We should also supply a default representation for the viewers that are unable to display swf figures, but this time it's not necessary to specify complicated options: just use the mode feature of ConT_EXt as in the following example

```

\startmode[Flash]
\externalfigure[smile.swf] [width=100px,height=100px]
\stopmode
\startnotmode[Flash]
\externalfigure[smile.png]
\stopnotmode

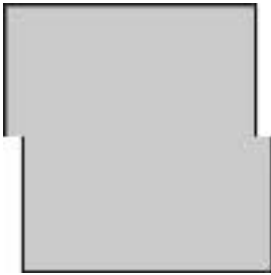
```

The same approach can be used to implement the start/stopSWFtoolsSCcode macros where the code is a swf script and the compiler is swfc (both are proprietary of swftools, see [5]):

```

\startSWFtoolsSCcode[name=action,
  compiler={/opt/luatex/minimals-2010-brejlov/tex/texmf-project/bin/swfc}]
.flash filename="action.swf" bbox=300x300 fps=50
.box mybox color=blue fill=green width=100 height=100
.put mybox
.frame 1
  .action:
    _root.angle += 0.05;
    mybox._x = 100*Math.cos(_root.angle)+100;
    mybox._y = 100*Math.sin(_root.angle)+100;
  .end
.frame 2
  .action:
    gotoFrame(0);
    Play();
  .end
.frame 3
.end
\stopSWFtoolsSCcode
\externalfigure[action.swf] [width=150px,height=150px]

```



Conclusion

From the point of view of a *traditional* (i.e. not \TeX) programmer \ConTeXt-mkiv has a neat approach for implementing the PDF specifications. The Lua language is small and complete, and the PDF specifications itself are clear enough: the problem arises with the rendering of the document. On average, a free PDF viewer other than AdobeReader has not the capability to show a RichMedia content and the printing of the pdf can be also problematic, so we **must** supply the correct alternative content at least with the `modes` mechanism. Following the same way of `start/stopSWFtoolsAScode` it is possible to implement a `start/stopFlexAS` code (see [6]) which is the preferable to the `swftools` compiler due some incompatibilities in the implementation of the ActionScript3 language.

References

- [1] Adobe: PDF Technology Center. Available at URL: <http://www.adobe.com/devnet/pdf/>
- [2] TIMO HARTMANN: The flashmovie package. Available at URL: <http://www.ctan.org/tex-archive/macros/latex/contrib/flashmovie>
- [3] LUIGI SCARSO: Playing with Flash in ConTEXt-mkiv. In The PracTEX Journal, Vol. 5, No. 1, 2010. ISSN 1556-6994. Available at URL: <http://www.tug.org/pracjourn/2010-1/scarso>
- [4] SWFTTools – SWF manipulation and generation utilities.
Available at URL: <http://www.swftools.org>
- [5] SWFC Manual. Available at URL: <http://www.swftools.org/swfc/swfc.html>
- [6] Adobe: Adobe Flex. Available at URL: <http://www.adobe.com/products/flex>

*luigi (dot) scarso (at) gmail (dot) com
Padova, Italy*

Abstract: MicroTalk is a short and technical paper that shows some unusual, hopefully useful, ideas following the schema “figure to code”. The main topic is always typographic programming in ConTeXt & Lua. A bit of Lua code, the `\clip` macro and Leptonica extensions are the ingredients for this recipe to cook a pdfsplit macro that take a pdf and try to split into parts as the `\vsplit` does with `\vboxes`.

Key words: Lua, Leptonica, PDF slicing, SWIG, MuPDF, Sumatra.

Abstrakt: Článek poukazuje na způsob nařezání PDF na proužky tak, jak je to známé pomocí příkazu `\vsplit` s `\vboxy`.

Klíčová slova: Lua, Leptonica, vysekávání z PDF, SWIG, MuPDF, Sumatra.

References

- [1] Leptonica. Available at URL: <http://www.leptonica.com/>
- [2] MuPDF – a lightweight PDF viewer and toolkit written in portable C. Available at URL: <http://www.mupdf.com/>
- [3] Sumatra PDF. Available at URL: http://en.wikipedia.org/wiki/Sumatra_PDF

luigi (dot) scarso (at) gmail (dot) com
Padova, Italy

1 Slice a pdf

Let's start with a bit of Lua code:

```
\startluacode
document.lscarsos = document.lscarsos or {}
function document.lscarsos.LuaSliceIt(Fig,H,W,L,FigOpt)
  local H = math.floor(string.gsub(H,"pt","")*2^16)
  local W = math.floor(string.gsub(W,"pt","")*2^16)
  local L = math.floor(string.gsub(L,"pt","")*2^16)
  local vh = 0
  local step = L
  local h = step
  local S = ""
  local FigOpt = FigOpt or ""
  while vh <= H do
    S =string.format("{\\clip[voffset=%dsp,
                    width=%dsp,
                    height=%dsp]
                    {\\externalfigure[%s][%s]}}
                    \\par\\nointerlineskip\\blank[1sp]",
                    vh,W,h,Fig,FigOpt)
    tex.sprint(tex.ctxcatcodes,S)
    vh = vh + step
  end
  if math.mod(H,step) > 0 then
    vh = vh - step + math.mod(H,step)
    S =string.format("{\\clip[voffset=%dsp,
                    width=%dsp,
                    height=%dsp]
                    {\\externalfigure[%s][%s]}}
                    \\par\\nointerlineskip\\blank[1sp]",
                    vh,W,math.mod(H,step),Fig,FigOpt)
    tex.sprint(tex.ctxcatcodes,S)
  end
end
\stopluacode
```

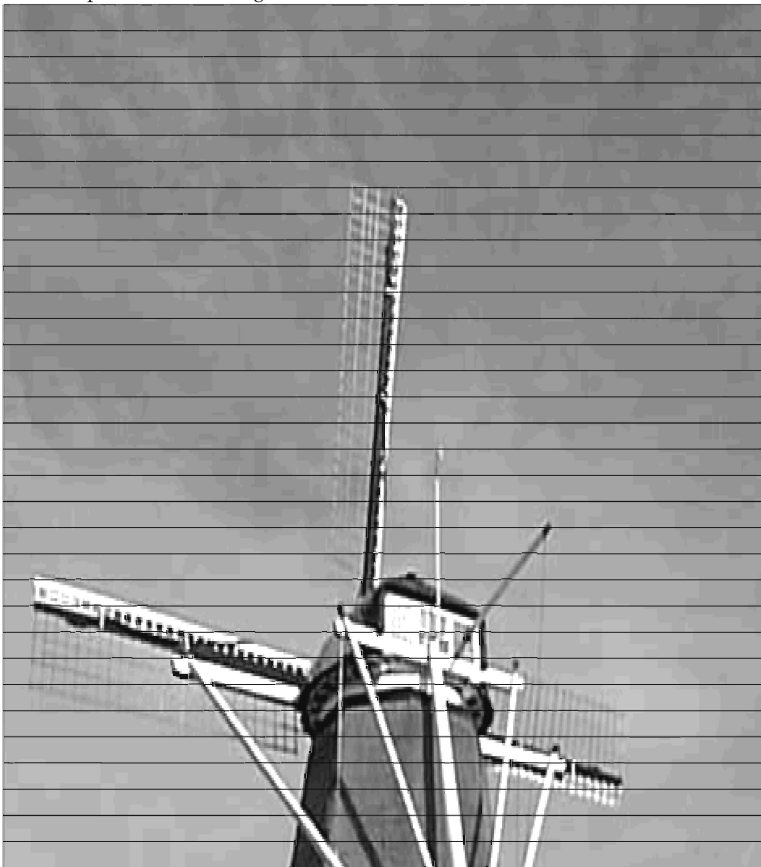
document.lscarsos.LuaSliceIt takes a pdf and “slices it” with slices of height L.

```

\bgroup
\newdimen\Hfig \newdimen\Wfig
\setbox1000=\hbox{\externalfigure[mill.png] [width=\textwidth,
                                             height=1.3\textheight]}
\Wfig=\wd1000 \Hfig=\dimexpr\ht1000+\dp1000\relax%
\ctxlua{document.lscarlo.LuaSliceIt("mill.png", "\the\Hfig",
                                     "\the\Wfig", "\the\dimexpr 1.0\lineheight\relax",
                                     "width=\\textwidth,height=1.3\\textheight")}
\egroup

```

It's not a problem with images:





But uniform slicing is wrong with text:

$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877344877345$
$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{1463919079740743966268954674710929768361083}{281670315928038407744716588098661706369472} = 5.197278507738630161795216686$
$\sum_{x=0}^{150} \frac{1}{x+1} = \frac{4195569667676135811153969815137073234944561746732919339914337201927}{749502901196827228266820481792118993292919127408542808267329424000} = 5.597803105200170187965715973$
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{18472920064106597929865025} = 3.173842337190749408690224740$
$\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914903558851799204586212$
$\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141592653589793121471200875$
$\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$
$\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141592653589793121471200875$

$\sum_{x=0}^{10} 2x+1$	
$4 \sum_{x=0}^{200000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$	

We can enumerate each slice, and observe that it depends only on the given step: with `step = 1.0` we have

$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345$	001
	002
$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{1463919079240743966260954674710929769361003}{281670315928038407744716598098661706369472} = 5.197278507738630161795216686$	003
	004
$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{4195569667676135811153669815137073234944561746732919339914337201927}{74950290119682728266820481792118993292919127408542808267329424000} = 5.597803105200170187965715973$	005
	006
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852384}{18472920064106597929865025} = 3.173842337190749408690224140$	007
	008
$\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914903558851799205886212$	009
	010
$\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141925075639790154273200075$	011
	012
$\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153992692$	013
	014
$\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141595986912015400462612518$	015
	016
$4 \sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$	017
	018
	019
	020

It's now clear that 002, 004, 006... are good break points while 001, 003, 005... must be avoided. if we collect all good break points we can then subdivide the pdf into the right `\vboxes`. These break points depend on `lineheight`, but we can choose another `step`, let's say 2mm, so that slices are independent from the body font:

```
\bgroup
\newdimen\Hfig \newdimen\Wfig
\setbox1000=\hbox{\externalfigure[pari_example.pdf][width=\textwidth]}
\Wfig=\wd1000 \Hfig=\dimexpr\ht1000+\dp1000\relax%
\ctxlua{document.lscarsolua.LuaSliceItNR("pari_example.pdf", "\the\Hfig",
"\the\Wfig", "\the\dimexpr 2mm\relax",
"width=\textwidth")}
\egroup
```

	001
$\sum_{x=0}^{10} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877345$	002
	003
$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{1463919079240743966260954674710929769361003}{281670315928038407744716598098661706369472} = 5.197278507738630161795216686$	004
	005
$\sum_{x=0}^{100} \frac{1}{x+1} = \frac{4195569667676135811153669815137073234944561746732919339914337201927}{74950290119682728266820481792118993292919127408542808267329424000} = 5.597803105200170187965715973$	006
	007
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = \frac{58630135791001973169852384}{18472920064106597929865025} = 3.173842337190749408690224140$	008
	009
$\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914903558851799205886212$	010
	011
$\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141925075639790154273200075$	012
	013
$\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153992692$	014
	015
$\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141595986912015400462612518$	016
	017
$4 \sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$	018
	019
	020

$\sum_{x=0}^{170} \frac{1}{x+1} = 4195569667676135811153969815137073234944561746732919339914337201927$	011
$\sum_{x=0}^{170} \frac{1}{x+1} = 749502901196827228266820481792118993292919127408542808267329424000$	012
$\sum_{x=0}^{170} \frac{1}{x+1} = 5.597803105200170187965715973$	013
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 58630135791001973169852204$	014
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 18472920064106597929865025 = 3.173842337190749408690224140$	015
$\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914983558851799204586212$	018
$\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141925875839790151271200075$	019
$\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$	023
$\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141595986912015488462612519$	025
$\sum_{x=0}^{3000000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$	026
	027
	028
	029
	030
	031
	032
	033
	034
	035
	036
	037
	038
	039
	040
	041
	042

We group together slices 1, 5, 9, 13, 14, 18, 22, 26, 31, 35, 39:

$\sum_{x=0}^{10} \frac{1}{x+1} = 83711$	001
$\sum_{x=0}^{10} \frac{1}{x+1} = 27720 = 3.019877344877344877344877345$	005
$\sum_{x=0}^{100} \frac{1}{x+1} = 1463919079240743966268954674710929768361083$	009
$\sum_{x=0}^{100} \frac{1}{x+1} = 281670315928038407744716588098661706369472 = 5.197278507738630161795216686$	012
$\sum_{x=0}^{150} \frac{1}{x+1} = 4195569667676135811153969815137073234944561746732919339914337201927$	013
$\sum_{x=0}^{150} \frac{1}{x+1} = 749502901196827228266820481792118993292919127408542808267329424000 = 5.597803105200170187965715973$	014
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 58630135791001973169852204$	018
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 18472920064106597929865025 = 3.173842337190749408690224140$	019
$\sum_{x=0}^{300} \frac{-1^x}{2x+1} = 3.144914983558851799204586212$	022
$\sum_{x=0}^{3000} \frac{-1^x}{2x+1} = 3.141925875839790151271200075$	026
$\sum_{x=0}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$	031
$\sum_{x=0}^{300000} \frac{-1^x}{2x+1} = 3.141595986912015488462612519$	035
$\sum_{x=0}^{3000000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$	039
	042

Now slicing works well:

$\sum_{x=0}^{10} \frac{1}{x+1} = 83711$	001
$\sum_{x=0}^{10} \frac{1}{x+1} = 27720 = 3.019877344877344877344877345$	005
$\sum_{x=0}^{100} \frac{1}{x+1} = 1463919079240743966268954674710929768361083$	009
$\sum_{x=0}^{100} \frac{1}{x+1} = 281670315928038407744716588098661706369472 = 5.197278507738630161795216686$	012
$\sum_{x=0}^{150} \frac{1}{x+1} = 4195569667676135811153969815137073234944561746732919339914337201927$	013
$\sum_{x=0}^{150} \frac{1}{x+1} = 749502901196827228266820481792118993292919127408542808267329424000 = 5.597803105200170187965715973$	014
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 58630135791001973169852204$	018
$\sum_{x=0}^{30} \frac{-1^x}{2x+1} = 18472920064106597929865025 = 3.173842337190749408690224140$	019

$4 \sum_{x=1}^{300} \frac{-1^x}{2x+1} = 3.144914903558851799204586212$	022
$4 \sum_{x=1}^{3000} \frac{-1^x}{2x+1} = 3.141925875839790151271200075$	026
$4 \sum_{x=1}^{30000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$	031
$4 \sum_{x=1}^{300000} \frac{-1^x}{2x+1} = 3.141595986912015488462612519$	035
$4 \sum_{x=1}^{3000000} \frac{-1^x}{2x+1} = 3.14159296923015460712643380$	039
	042

```

\bggroup
\newdimen\Hfig \newdimen\Wfig
\setbox1000=\hbox{\externalfigure[pari_example.pdf][width=\textwidth]}
\Wfig=\wd1000 \Hfig=\dimexpr\ht1000+\dp1000\relax%
\startluacode
document.lscarlo.GoodPoints = {1,5,9,13,14,18,22,26,31,35,39}
\stoptluacode
\ctxlua{document.lscarlo.LuaSliceItAndCollect("pari_example.pdf",
"\the\Hfig", "\the\Wfig", "\the\dimexpr 2mm\relax",
"width=\textwidth", "")}
\egroup

function document.lscarlo.LuaSliceItAndCollect(Fig,H,W,L,FigOpt)
local H = math.floor(string.gsub(H,"pt","")*2^16)
local W = math.floor(string.gsub(W,"pt","")*2^16)
local L = math.floor(string.gsub(L,"pt","")*2^16)
local vh = 0
local step = L
local h = step
local S = ""
local FigOpt = FigOpt or ""
local NR = 0
local Payload = "{\\ruledhbox{\\clip[voffset=%dsp,width=%dsp,height=%dsp]
{\\externalfigure[\\s][\\s]}\\llap{\\tfx%03d}}}"
\\par\\nointerlineskip\\blank[1sp]"
local GoodPoint = document.lscarlo.GoodPoints or {}
local j = 1
local Vboxes = {}
local prevvh = 0
while vh < H do
NR =NR +1
vh = vh + step
if GoodPoint[j] == NR then
Vboxes[#Vboxes+1] = {from=prevvh,to=vh,mark=NR}
--tex.sprint(tex.ctxcatcodes, "(" ,prevvh, ", ",vh, ") ")
end
j = j + 1
end

```

```

    prevvh = vh
    j = j + 1
end
end
if math.mod(H,step) == 0 then H = H + 1 end
if math.mod(H,step) > 0 then
    NR = NR + 1
    vh = vh - step + math.mod(H,step)
    Vboxes[#Vboxes+1] = {from=prevvh,to=vh,mark=NR}
end
Payload = "\\ruledvbox{\\hbox{\\clip[voffset=%dsp,width=%dsp,height=%dsp]
{\\externalfigure[\\s][\\s]}\\llap{\\tfx\\%03d}}
\\par\\nointerlineskip\\blank[1sp]"
for i,v in ipairs(Vboxes) do
    S =string.format(Payload,v.from,W,v.to-v.from,Fig,FigOpt,v.mark)
    tex.sprint(tex.ctxcatcodes,S)
end
end

```

It's now rather trivial to modify this macro to build a `\vbox` to be used with `\vsplit`.

2 Leptonica

Leptonica (<http://www.leptonica.com>) is a pedagogically-oriented open source site containing software that is useful for image processing and image analysis applications. It's not difficult to build a binding for Lua with SWIG. To use Leptonica we must first convert the pdf in a black and white bitmap with

```
pdftoppm -mono -r 72 -f 1 -l 1 pari_example.pdf pari_example
```

(this saves page 1 in `pari_example-000001.pbm` with a resolution of 72dpi). Then we scan the bitmap with `lept_get_breaks` searching for a white row and we store the y coordinates in `GoodPoints`. In the end `LuaCollect` builds the appropriate vboxes.

```
\bgroup
\newdimen\Hfig \newdimen\Wfig
\setbox1000=\hbox{\externalfigure[pari_example.pdf][width=\textwidth]}
\Wfig=\wd1000\relax\Hfig=\dimexpr\ht1000+\dp1000\relax%
\executesystemcommand{pdftoppm -mono -r 72 -f 1 -l 1 pari_example.pdf
pari_example}
\ctlua{document.lscarlo.GoodPoints =
    document.lscarlo.lept_get_breaks("pari_example-000001.pbm",72)}
\ctlua{document.lscarlo.LuaCollect("pari_example.pdf",
    "\the\Hfig", "\the\Wfig", [[width=\textwidth]])}
\egroup
```

Let's see what happens:

$\sum_{x=1}^{100} \frac{1}{x+1} = \frac{83711}{27720} = 3.019877344877344877344877344$
$\sum_{x=1}^{1000} \frac{1}{x+1} = \frac{1463919079240743966268954674710929760361003}{281670315928038407744716588098661706369472} = 5.197278507738630161795216686$
$\sum_{x=1}^{10000} \frac{1}{x+1} = \frac{4195569667676135811153969815137073234944561746732919339914337201927}{749502901196827228266820481792118993292919127408542808267329424000} = 5.597803105200170187965715973$
$4 \sum_{x=1}^{100} \frac{-1^x}{2x+1} = \frac{58630135791001973169852284}{10472920064106597929865025} = 3.173842337190749408690224140$
$4 \sum_{x=1}^{1000} \frac{-1^x}{2x+1} = 3.144914903558851799204586212$
$4 \sum_{x=1}^{10000} \frac{-1^x}{2x+1} = 3.141925875839790151271200075$
$4 \sum_{x=1}^{100000} \frac{-1^x}{2x+1} = 3.141625985812043238153993692$
$4 \sum_{x=1}^{1000000} \frac{-1^x}{2x+1} = 3.141595986912015488462612519$
$4 \sum_{x=1}^{10000000} \frac{-1^x}{2x+1} = 3.141592986923015460712643380$

A good breakpoint is marked with an horizontal rule: a black stripe means that good points are very tight there.

```
require("leptonica")
function document.lscarso.lua_pixGetPixel(pixs,x,y)
    local scratch = 0
    local res = ''
    scratch = leptonica.uti_getref_l_uint32()
    if (leptonica.pixGetPixel(pixs,x,y,scratch) == 0) then
        res = leptonica.uti_valref_l_uint32(scratch)
    else
        print("! Error on ",x,y)
        res = ''
    end
    return res
end

function document.lscarso.lept_get_breaks(filename,res)
    -- os.execute([[pdftoppm -mono filename]])
    local mainName = filename
    local pixs = leptonica.pixRead(mainName)
    local lua_pixGetPixel = document.lscarso.lua_pixGetPixel
    local GoodPoint = {} -- document.lscarso.GoodPoints or {}
    if pixs then
        local w = leptonica.pixGetWidth(pixs)
        local h = leptonica.pixGetHeight(pixs)
        local d = leptonica.pixGetDepth(pixs)
        local out = ''
        local bit
        local storey = true
        local go
        GoodPoint[1] = 0
        for y=0,h-1 do
            out = ''
            storey = true
            for x=0,w-1 do
                bit = lua_pixGetPixel(pixs,x,y)
                if bit == 1 then
                    storey = false
                    break
                end
            end
            end
            if storey and (math.floor(0.5+ (y/res) *72.27*2^16)
                ~= GoodPoint[#GoodPoint]) then
```

```

        GoodPoint[#GoodPoint +1] = math.floor(0.5+ (y/res)
*72.27*2^16)
        end
        storey = true
    end
    GoodPoint[#GoodPoint +1] = math.floor(0.5+ (h/res) *72.27*2^16)
else
    tex.sprint("ERROR")
end
return GoodPoint
end

function document.lscarsolua.LuaCollect(Fig,H,W,FigOpt)
    local H = math.floor(string.gsub(H,"pt","")*2^16)
    local W = math.floor(string.gsub(W,"pt","")*2^16)
    local vh = 0
    local h = step
    local S = ""
    local FigOpt = FigOpt or ""
    local NR = 0
    local Payload = ""
    local GoodPoint = document.lscarsolua.GoodPoints or {}
    local j = 1
    local Vboxes = {}
    local prevvh = 0
    local truey = GoodPoint[#GoodPoint]
    local y_ratio = H / truey
    for i,v in ipairs(GoodPoint) do
        if GoodPoint[i+1] == nil then
            break
        end
        NR = NR +1
        Vboxes[#Vboxes+1] = {from=v,to=GoodPoint[i+1],mark=NR}
    end
    Payload = "\\ruledvbox{\\hbox{\\clip[voffset=\\%dsp,width=\\%dsp,height=\\%dsp]
        {\\externalfigure[\\%s][\\%s]}}}\\par\\nointerlineskip\\blank[1sp]"
    for i,v in ipairs(Vboxes) do
        S =string.format(Payload,y_ratio*v.from,W,y_ratio*(v.to-v.from),Fig,FigOpt)
        tex.sprint(tex.ctxcatcodes,S)
    end
end
end

```

3 MuPDF

We can replace the `pdftoppm` call by building a Lua binding to MuPDF, “a light-weight PDF viewer and toolkit written in portable C” (<http://www.mupdf.com>). Sumatra (http://en.wikipedia.org/wiki/Sumatra_PDF) is a fast pdfviewer based on MuPDF.

Despite MuPDF is a well written library and there are several examples that explain how to use it, it's not easy to write a converter.

What follow is only a part of an almost literal translation into Lua of the C program `pdfdraw.c` and even so the debugging is difficult.

```
function drawpbm(pagenum)
    local xref = pdfdraw.targetpdf.xref
    local drawrotate = pdfdraw.drawrotate
    local drawzoom = pdfdraw.drawzoom
    local drawbands = pdfdraw.drawbands
    local drawpattern= pdfdraw.drawpattern
    drawloadpage(pagenum)
    local drawpage = pdfdraw.drawpage
    local drawcache = pdfdraw.drawcache
    local bbox,w,h, bh
    local pix
    local fd,name
    name = pdfdraw.outname
    local ctm = mupdf.fz_identity()
    ctm = mupdf.fz_concat(ctm, mupdf.fz_translate(0, - drawpage.mediabox.y1))
    ctm = mupdf.fz_concat(ctm, mupdf.fz_scale(drawzoom, - drawzoom))
    ctm = mupdf.fz_concat(ctm, mupdf.fz_rotate(drawrotate + drawpage.rotate))
    bbox = mupdf.fz_roundrect(mupdf.fz_transformrect(ctm, drawpage.mediabox))
    w = bbox.x1 - bbox.x0;
    h = bbox.y1 - bbox.y0;
    bh = h / drawbands;
    if NotNil(drawpattern) then
        fd = io.open(name,'wb')
        if (fd == nil) then
            die(fz_throw("ioerror: could not create raster file '%s'", name));
        end
        fd:write(string.format("P4\n%d %d\n", w, h));
    end
    pix = mupdf.fz_newpixmap(mupdf.pdf_devicergb, bbox.x0, bbox.y0, w, bh)
    mupdf.fz_clearpixmap(pix, 0xFF);
    for b = 0, drawbands-1 do
        local dev, error
        dev = mupdf.fz_newdrawdevice(drawcache, pix)
```

```

error = mupdf.pdf_runcontentstream(dev, ctm, xref, drawpage.resources, drawpage.contents)
if (error>0) then
    die(fz_rethrow(error, "cannot draw page %d in PDF file '%s'", pagenum, basename))
end
mupdf.fz_freedevice(dev)
if NotNil(drawpattern) then
    for y = 0, pix.h -1 do
        local column = y * pix.w * 4
        local dst = {}
        local bit = 0
        local r = 0
        for x = 0, pix.w-1 do
            local v = (1+(mupdf.uti_samples_arr_getitem(pix.samples,(x * 4 + 1) +
column)))*77
            v = v + (1+(mupdf.uti_samples_arr_getitem(pix.samples,(x * 4 + 2) + column)))*150
            v = v + (1+(mupdf.uti_samples_arr_getitem(pix.samples,(x * 4 + 3) + column)))*28
            v = math.floor(v/256)
            local d = 1
            if v > 200 then d = 0 end
            r = r + (d*2^(7-bit))
            bit = bit +1
            if bit == 8 then
                bit = 0
                dst[#dst+1] = string.char(r)
                r = 0
            end
            if bit > 0 then
                dst[#dst+1] = string.char(r)
            end
            fd:write(table.concat(dst))
        end
        pix.y = pix.y + bh;
        if (pix.y + pix.h > bbox.y1) then
            pix.h = bbox.y1 - pix.y;
        end
    end
end
mupdf.fz_droppixmap(pix)
if NotNil(drawpattern) then
    fd:close()
end
drawfreepage()
print()
end

```

The complete binding is truly more difficult to debug: even if SWIG does an excellent work, it's necessary to manage the details of implementation of the library. In this case `pdftopbm` does already an excellent work, it's well tested, stable and ready to use; the call of an external program is not so expensive compared to a plug in.

4 Conclusion

Lua code is much nearer to plain standard Lua: ConTeXt offers some shortcuts that is better to learn. For example `math.floor(string.gsub("10pt", "pt", "")*2^16)` can be replaced by `string.todimen("10pt")` and possibly other things related to print.

The algorithm that finds a good breakpoint is rather simple: for example the are some unwanted good breakpoints between a Σ and the its subscript. This leads to break an object that should not be broken even if it has some white rows inside. A practical solution is to consider a line with thickness grater than 1 pixel (but it's better to use metric dimensions).

These ideas are also valid for scanned text but then we must take care of noise (and Leptonica can help a lot here).

Finally, it's better to consider carefully the opportunity of a binding. As shown with MuPDFD, sometimes the traditional way is still the best way.

About μ Talk

μ Talk is a short and technical paper that shows some unusual, hopefully useful, ideas following the schema "figure \rightarrow code". The main topic is always typographic programming in ConTeXt & Lua.

Experiences Typesetting OpenType Math with Lua \LaTeX and Xe \LaTeX

Zkušenosti se sazbou matematiky ve formátu OpenType math v Lua \LaTeX u a Xe \LaTeX u

ULRIK VIETH

Abstract: When Lua \TeX first provided support for OpenType math typesetting in version 0.40, high-level macro support for math typesetting was first developed for Con \TeX t MkIV, while support for Lua \LaTeX was initially limited to a very low-level or non-existent. In the meantime, this gap has been closed by recent developments on macro packages such as luaotfload, fontspec, and unicode-math, so \LaTeX users are now provided with a unified high-level font selection interface for text and math fonts that can be used equally well with both Lua \LaTeX and Xe \LaTeX . While a unified high-level interface greatly improves document interchange and eases transitions between systems, it does not guarantee that identical input will always produce identical output on different engines, as there are significant differences in the underlying implementations of math typesetting algorithms. While Lua \TeX provides a full-featured implementation of OpenType math, Xe \TeX has taken a more limited approach based on a subset of OpenType parameters to provide the functionality of traditional \TeX engines.

Given the possibility of running exactly the same test files on both engines, it now becomes feasible to study those differences in detail and to compare the results. Hopefully, this will allow to draw conclusions how the quality of math typesetting is affected and could be improved by taking advantage of a more sophisticated, full-featured OpenType math implementation.

Key words: Lua \LaTeX , Xe \LaTeX , OpenType math, math typesetting, fontspec package, Cambria, Asana, XITS, Neo Euler.

Abstrakt: Jelikož Lua \TeX podporuje Open Type math až od verze 0.40, byla podpora matematické sazby na vyšší úrovni vytvořena nejprve pro Con \TeX t MkIV, zatímco podpora pro Lua \LaTeX byla nízká nebo nebyla žádná. Další vývoj však tuto mezeru zacelil – uživatelé \LaTeX u mají nyní k dispozici jednotné rozhraní pro připojení fontů pro běžný text i pro matematickou sazbu pomocí balíčků luaotfload, fontspec a unicode-math;

obojí lze celkem stejně dobře využít v Lua \LaTeX u i v Xe \LaTeX u. I když toto jednotné rozhraní značně zjednodušuje výměnu dokumentů i přenos mezi různými systémy, nezaručuje, že tentýž vstup vytvoří vždy tentýž výstup na různých počítačích kvůli významným odlišnostem v implementaci algoritmů pro matematickou sazbu. Zatímco Lua \TeX poskytuje úplnou implementaci všech vlastností OpenType math, Xe \TeX převzal jen část z nich s ohledem na tradiční implementace \TeX u.

Maje možnost překládat stejné testovací soubory v obou implementacích, bylo možné podrobně zkoumat jejich rozdíly a porovnat výslednou matematickou sazbu. Doufáme, že toto přispěje k zjištění, co ovlivňuje kvalitu matematické sazby a jak ji zlepšit implementací výhod kompletního formátu OpenType math.

Klíčová slova: Lua \LaTeX , Xe \LaTeX , formát OpenType math, sazba matematiky, balíček fontspec, Cambria, Asana, XITS, Neo Euler.

Lua \TeX math

$$\Delta \mathbf{E} - \frac{1}{c^2} \frac{\partial^2 \mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0} \nabla \lambda + \mu_0 \frac{\partial \mathbf{j}}{\partial t},$$

$$\Delta \mathbf{B} - \frac{1}{c^2} \frac{\partial^2 \mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot} \mathbf{j}.$$

```
% !TeX program = lualatex
\documentclass[fleqn]{article}
\usepackage{fontspec, unicode-math}
\setromanfont{Cambria}
\setmathfont{Cambria Math}
\begin{document}
\input{luatex-fixes}
\input{math-test}
\end{document}
```

Xe \TeX math

$$\Delta \mathbf{E} - \frac{1}{c^2} \frac{\partial^2 \mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0} \nabla \lambda + \mu_0 \frac{\partial \mathbf{j}}{\partial t},$$

$$\Delta \mathbf{B} - \frac{1}{c^2} \frac{\partial^2 \mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot} \mathbf{j}.$$

```
% !TeX program = xelatex
\documentclass[fleqn]{article}
\usepackage{fontspec, unicode-math}
\setromanfont{Cambria}
\setmathfont{Cambria Math}
\begin{document}
\input{xetex-fixes}
\input{math-test}
\end{document}
```

Can you spot the difference?

$$\Delta \mathbf{E} - \frac{1}{c^2} \frac{\partial^{\mathbb{Z}} \mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0} \nabla \lambda + \mu_0 \frac{\partial \mathbf{j}}{\partial t},$$

$$\Delta \mathbf{B} - \frac{1}{c^2} \frac{\partial^{\mathbb{Z}} \mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot} \mathbf{j}.$$

```
% !TeX program = pdflatex
\documentclass{article}
\usepackage{pdfpages}
\begin{document}
\includepdfmerge
[nup=2x1, noautoscale=true, delta=-21cm 0]
{xelatex-test.pdf, 1, lualatex-test.pdf, 1}
\end{document}
```

Introduction

In this paper, we will review the state of recent developments of new \TeX engines and corresponding macro packages to support math typesetting with Unicode and OpenType math fonts, based on our experiences from testing the \TeX Live 2010 pretest distribution [1].

In the first part, we will summarize the available choices of \TeX engines and macro packages, as well as the available choices of math fonts, which can be used for testing OpenType math typesetting.

In the second part, we will report our experiences testing the various \TeX engines, macro packages and fonts, and we will report our findings which kind of problems were encountered and how these problems were resolved or circumvented.

In the third part, we will analyze and compare the results of running identical documents through different \TeX engines using different implementations of math typesetting algorithms, and we will try to draw conclusions how the quality of math typesetting is affected and could be improved.

Reviewing the state of OpenType math support in \TeX Live 2010

In recent years, many developments related to new \TeX engines and corresponding macro packages and fonts have focused on providing support for Unicode and OpenType, not just for text typesetting, but also for math typesetting (which is still important as one of the traditional strong-holds of \TeX).

Unicode and OpenType math technology

While the pre-history of Unicode text support dates back to the mid-1990s, most activities related to Unicode math support only became possible during the last few years, after a suitable font technology was developed as an extensions to the OpenType font format.

When the efforts to bring math into Unicode were started in the late 1990s by a consortium of scientific publishers, the original focus was on identifying math symbols and getting them accepted into Unicode [2, 3]. Once this was done, the focus shifted to developing a reference font implementation, the so-called STIX fonts, to provide the necessary glyph shapes [4].

While the STIX project was spending nearly a decade waiting for fonts to be designed, the lack of a suitable font technology for math typesetting was overlooked for a long time. While traditional \TeX font formats supported math typesetting in their own way, they suffered from limitations and questionable design decisions [5].

On the other hand, mainstream font formats such as OpenType did not provide any support for the semantics of math typesetting.

Ironically, the lack of a suitable math font technology was only resolved when Microsoft started to develop support for MS Office 2007. Given their influence as a vendor controlling the OpenType font specification [6], they simply went ahead and created an extension of the OpenType font format containing a MATH table [7] and commissioned the design of Cambria Math as a reference implementation of an OpenType math font [8, 9]. In addition, they also developed a simplified math input language known as 'linear math' [10].

While there are sometimes strong reservations about accepting a vendor-defined file format, especially an unreleased one coming from Microsoft, developers of font tools such as FontForge [11] as well as developers of \TeX engines were willing to accept OpenType math as a *de facto* standard, because it filled a need and also because it turned out to be well-designed.

Upon closer analysis, many concepts of OpenType math could be seen as obvious extensions or generalizations of traditional concepts of math typesetting in \TeX [12]. Moreover, most OpenType math font parameters could be identified to have a direct correspondence to \TeX math font parameters [13], which had also been carefully analyzed in previous studies [14, 15].

OpenType math support in \TeX engines

When \XeTeX first added OpenType math support in version 0.997 as of 2007 [16], it kept \TeX 's traditional math typesetting algorithms essentially unchanged and only used a small subset of OpenType font parameters to initialize the required \TeX font parameters.

When \LuaTeX also added OpenType math support in version 0.40 as of 2009 [17, 18, 19], it introduced a number of extensions and generalizations to \TeX 's math typesetting algorithms, aiming to provide a full-featured implementation of OpenType math.

As of \TeX Live 2010, both new \TeX engines supporting Unicode and OpenType math typesetting have been added to the mainstream distributions and have become widely available for using and testing the new features. However, their acceptance also depends on providing adequate macro package and font support.

OpenType math support in macro packages

When \XeTeX was first added to \TeX distributions, it was easily accessible to \LaTeX users with \XeLaTeX . A high-level font selection interface for text fonts was developed with the font spec package [20, 21], which became widely used as a standard package for \XeLaTeX .

When XeTeX added math support, a corresponding high-level font selection interface for math fonts was also developed with the `unicode-math` package [22], but unlike `fontspec` it wasn't released until recently.

When LuaTeX added math support, high-level macro support was initially developed for ConTeXt MkIV [23], while macro support for LuaLaTeX (or Plain LuaTeX) was initially limited to the `luaotfload` package [24], which provided only a low-level interface.

As of TeX Live 2010, LaTeX macro package support for Unicode and OpenType math typesetting has been much improved, as both the `fontspec` and `unicode-math` packages have undergone a complete rewrite, adding support for LuaLaTeX (based on `luaotfload`) to the code originally developed for XeLaTeX.

As a result, LaTeX users are now provided with a unified high-level font selection interface for text and math fonts [25] that can be used equally well with both XeLaTeX and LuaLaTeX.

Given this interface, selecting a different math font (such as Cambria Math) can be as easy as this:

```
\usepackage{fontspec,unicode-math}
\setmainfont[<options>]{Cambria}
\setmathfont[<options>]{Cambria Math}
```

Using the options of `\setmathfont`, a number of details of math typesetting can be easily configured, including the behavior of math alphabets (such as upright vs. italic for normal and bold, uppercase and lowercase, Latin and Greek), which will make it much easier to support the specific requirements of math typesetting in various fields of sciences [26].

OpenType math fonts

Regardless of font technology, developing math fonts has always been far from easy and choices of math fonts have always been severely limited. In this respect, the situation of OpenType fonts today is not much different from the situation of PostScript fonts in the 1990s. While there are countless choices of text fonts, there are only very few math fonts available, and even fewer of them are freely available.

As of mid-2010, we have the following choices of OpenType math fonts at our disposal:

- Cambria Math [8], the original reference math font, commissioned by Microsoft for Office 2007,
- Asana Math [27], a Palatino-like math font derived from a repackaging of the `mathpazo` fonts,
- XITS Math [28], a Times-like math font derived from a repackaging of the STIX fonts [29],
- Neo Euler [30], an upright math font derived from Hermann Zapf's redesign of AMS Euler fonts [31].

Except for Cambria Math, all of these fonts are freely available, either from CTAN (if already released) or from GitHub (if still under development).

As of TeX Live 2010, Asana Math and XITS Math are both included in the distribution, but Cambria Math and Neo Euler have to be obtained separately and need to be installed manually in your `texmf-local` tree.

Once installed, each of the fonts can be used in the same way, but the range of symbols and math alphabets available may differ significantly between fonts.

Experiences testing OpenType math support in TeX Live 2010

Testing the functionality and quality of OpenType math typesetting implies testing a complex system, consisting of typesetting engines, macro packages and fonts (with embedded intelligence), which have to interact properly to produce the desired results.

Given the inherent complexity, there are a large number of problems which can occur, and most likely will occur, so we have to consider the possibilities of engine problems, macro problems, font problems and font loading issues.

Problems with TeX engines

Problems with TeX engines can be of several different kinds, ranging from fatal ones (such as unexpected crashes or malfunctions) to more subtle ones (such as hidden bugs in the math typesetting algorithms producing incorrect results).

Traditionally, TeX engines have enjoyed a reputation of being extremely robust and totally free of bugs. Unfortunately, such expectations no longer hold true when it comes to new TeX engines, which are under active development and which don't have the luxury of two decades of time to eliminate all possible bugs.

Engine problems of the fatal kind are therefore a very real possibility, which may even prevent or delay further testing until the problems can be resolved.

While testing the TeX Live 2010 pretest distribution, a number of problems were encountered for XeTeX on some 64-bit Linux platforms, resulting in crashes or malfunctions upon loading OpenType math fonts, which are likely to be caused by incompatibilities with external library dependencies.

Unfortunately, there was not enough time to debug these problems before the deadline for the TeX Live 2010 binaries, so the problems remain unresolved for now and need to be revisited eventually. As a workaround, it may be possible to use 32-bit binaries on 64-bit Linux platforms, which do not exhibit such problems.

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0,$$

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0.$$

Figure 1: Comparison of the size of delimiters in Asana Math for Lua \TeX 0.60.x and Lua \TeX 0.61. Due to a bug in the math typesetting algorithms, the extensible version of delimiters was applied too soon. (The example shows the Dirac equation from relativistic quantum mechanics.)

Engine problems of the more subtle kind were found in Lua \TeX , when a bug was discovered for some math fonts (such as Asana Math), which resulted in applying the extensible version of delimiters before exhausting all available sizes of big delimiters. (An example of the incorrect behavior is illustrated in Figure 1.)

In the meantime, this bug has already been fixed in Lua \TeX 0.61, but again it was too late to include the fix in \TeX Live 2010 which still uses Lua \TeX 0.60.x.

Problems with OpenType font metrics

Problems with OpenType fonts can also be of different kinds, ranging from fatal ones (such as containing malformed data structures causing \TeX engines to crash) to more subtle ones (such as providing incorrect values of font metric parameters causing \TeX engines to produce incorrect results). In addition, font problems can also include encoding issues or incorrect glyph shapes.

Font problems of the fatal kind did not occur while testing OpenType math fonts, but a similar kind of problem was recently reported for some OpenType text fonts. The problem was quickly addressed with a fix in Lua \TeX 0.61 to make the font parsing algorithms more robust about handling unexpected values.

Font problems of the more subtle kind were found with incorrect parameter settings in the MATH table of Cambria Math and Asana Math, which caused Lua \TeX to produce incorrect results.

The problem is related to the OpenType math parameter `DisplayOperatorMinHeight`, which is used in Lua \TeX to determine the minimum size of displaystyle operators. If this parameter is incorrectly set too small in the font, displaystyle operators will appear in the same size as textstyle operators. (An example of the incorrect behavior is illustrated in Figure 2.)

As it turned out, the problem had already been found earlier when OpenType math support in Lua \TeX was first tested with Con \TeX t, and a workaround had been applied, but the same problem now reappeared when Lua \TeX was tested with LuaLa \TeX .

$$\int_F \varepsilon_0 \mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0,$$

$$\int_F \varepsilon_0 \mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0.$$

Figure 2: Comparison of the size of displaystyle operators in Cambria Math for Lua \TeX with incorrect parameter values of `DisplayOperatorMinHeight` and with a correction applied at the macro level. (The example shows the integral form of the Maxwell equations from electrodynamics.)

In Con \TeX t, a patch for incorrect font parameters had been applied in the font loading code at the Lua level in `font-pat.lua`, but a similar patch was missing in `luaotfload`. Until this is fixed, a workaround to the same effect can be applied at the macro level by setting `\Umathoperatorssize\displaystyle=13.6pt`.

In any case, such kinds of patches for specific font parameter values only present a stop-gap solution until the actual fonts can be fixed. Whether or not this will be possible, critically depends on the cooperation of the font developer or distributor and may range between very quickly (for some open source projects) and next to impossible (for some commercial fonts).

Problems with OpenType font shapes

Font problems of yet another kind can occur when the assignment of glyph shapes to Unicode slots does not match the expectations, or when an incorrect font style is used for some glyphs.

One such problem was discovered for the partial differential symbol in Cambria Math and XITS Math. Since Unicode math provides a separate slot for a math italic version (U+1D715), one would expect the default slot (U+2202) to be reserved for the upright version, yet the Unicode font tables incorrectly happen to show an italic version in both slots and no upright version.

Given the confusion in the Unicode font tables, it is not surprising that several OpenType math fonts have inherited the same problem. Unfortunately, such font problems are unlikely to be fixed anytime soon.

∂	∂	∂	∂	Cambria Math
∂	∂	∂	∂	XITS Math
∂	∂	∂	∂	Asana Math

Figure 3: Comparison of different font shapes of the partial differential symbol (upright, italic, bold upright, bold italic) as they appear in Cambria Math, XITS Math, and Asana Math. Besides the confusion of upright vs. italic there are also some obvious problems for some of the bold italic versions.

Problems with \TeX macro packages

Problems with \TeX (or Lua) macro packages are usually easy to fix. In the course of the \TeX Live 2010 pretest, a number of such issues were found in `luaotfload` and `unicode-math`, which have already been fixed.

Only one issue has remained unresolved, which is related to the use of the `\hbar` macro. In a traditional \LaTeX setting, `\hbar` is a macro which overlays the italic letter *h* with a bar accent from `cmr` to produce \hbar . By contrast, `\hslash` is a macro to access a ready-made glyph from the AMS fonts to produce \hbar .

In a Unicode math setting, only `\hslash` is assigned to a slot in an OpenType math font (U+210F), while there is no equivalent assignment for `\hbar`, which has somehow retained its original macro definition and still uses a glyph from `cmr` to create the overlay. For lack of a better solution, it would be better to define `\hbar` as an alias for `\hslash` in `unicode-math`.

Quite a different effect occurs in \ConTeXt , where `\hbar` is interpreted as a diacritic text character (*h*) in upright shape (U+0127), which may be appropriate in text typesetting, but not necessarily in a math formula. As in `unicode-math`, it would be better to define `\hbar` as an alias for `\hslash` in \ConTeXt as well.

Problems caused by interactions between \TeX macro packages and \TeX engines

Yet another kind of problem was discovered recently, which was caused by an interaction problem between macro packages and \TeX engines, specifically between the `unicode-math` package and the \XeTeX engine.

As it turned out, `unicode-math` allocated a new math family for the OpenType math font (such as family 4) while \XeTeX (unlike \LuaTeX) somehow still expected certain math font parameters to be taken from families 2 and 3 (as in traditional \TeX engines).

As a result, the preloaded font metric parameters from `cmsy` and `cmex` were incorrectly used to determine the spacing of math instead of the font parameters from the OpenType math font.

A fix for this problem is still pending, but most likely it would involve changing the `unicode-math` package to account for different engine-specific behaviors of \LuaTeX and \XeTeX . As a workaround, we can apply a fix by reassigning the fonts in families 2 and 3 after loading an OpenType math font in family 4:

```
\ifxetex\everymath{
  \textfont3 = \textfont4
  \textfont2 = \textfont4
  \scriptfont2 = \scriptfont4
  \scriptscriptfont2 = \scriptscriptfont4
}\fi
```

Font-loading problems

Font-loading problems are usually easy to fix or avoid, once the cause of the problem has been understood. Nevertheless, such kinds of problems present a frequent source of frustration for unwary users, so it may well be useful to discuss our experiences regarding the font loading problems we encountered in the course of testing OpenType math with \TeX Live 2010.

What is important here is to understand that different mechanisms are used to locate OpenType fonts in different \TeX engines and macro packages.

In \XeTeX , the `fontconfig` library is used to locate OpenType fonts, and this mechanism applies equally well for system fonts installed in the system font path as for fonts installed in your \TeX Live distribution.

Depending on your installation, it may be necessary to adjust the `fonts.conf` config file to include the font directories in your `texmf-dist` or `texmf-local` tree, and to refresh the font cache with the `fc-cache` command. Once a font directory has been added to the search path, all kinds of font files will be found there, regardless of where the font files are located.

In \LuaTeX , the `kpathsea` path searching library is used to locate fonts, which depends on the assignment of file extensions (such as `*.ttf` or `*.otf`) to different search paths. As a result of this, `cambrtia.ttc` will only be found in the `fonts/truetype` tree, while `euler.otf` will only be found in the `fonts/opentype` tree.

In addition to that, \ConTeXt and `luaotfload` on \LuaTeX use yet another font-loading mechanism based on a file cache implemented in Lua, which circumvents the `kpathsea` library completely. System fonts outside the `TEXMF` tree will be located using the `fonts.conf` config file to look up the font directories, but without using the `fontconfig` library. Once a font directory has been added to the file cache, all kinds of font files will be found there, regardless of where the fonts are located, similar to the `fontconfig` library.

Comparing and testing the quality of OpenType math typesetting

To proceed with a study the quality of OpenType math font support as of \TeX Live 2010, we have the following choices of typesetting engines and macro packages at our disposal (disregarding Plain \LuaTeX and \XeTeX which only provide low-level support):

- \LuaTeX with \ConTeXt MkIV
- \LuaTeX with \LuaLaTeX
- \XeTeX with \XeLaTeX

While both LuaLaTeX and ConTeXt share the same TeX engine and the same implementation of math typesetting algorithms, they differ in their high-level user interface and also in the intermediate levels of font loading code (such as luaotfload).

While both LuaLaTeX and XeLaTeX share the same user interface of unicode-math and fontspec, they are based on different TeX engines, LuaTeX and XeTeX, which differ significantly in their implementations of math typesetting algorithms.

Comparing the results of LuaLaTeX and ConTeXt should not be expected to expose any differences in the output from identical math typesetting algorithms, but if there are any differences, a closer analysis should help to detect bugs or inconsistencies in the different macro packages and/or in the font loading code.

Comparing the results of LuaLaTeX and XeLaTeX, however, should indeed be expected to expose some differences in the underlying typesetting algorithms. Hopefully, an analysis of these differences should allow to draw conclusions how the quality of math is affected and could be improved by taking advantage of a full-featured implementation of OpenType math.

Testing a sampling of OpenType math

When we began testing OpenType math support with TeX Live 2010, we didn't have time to do systematic and comprehensive testing, which would have been a very time-consuming and tedious task.

Instead, we wanted to get some quick impressions how well OpenType math support worked and if it would be ready for practical use, so we concentrated on testing just a sampling of mathematical notations. Given our personal background in typesetting mathematical physics, we started by creating a sample test document containing a selection of famous equations from various fields of physics, sampling various kinds of mathematical notations.

In addition to testing the available choices of TeX engines and macro packages, we also wanted to test a sampling of the available OpenType math fonts, so we proceeded to typeset identical copies of our test files with different TeX engines and with different choices of math fonts for each of Cambria Math, XITS Math, Asana Math, and Neo Euler.

Some examples of typesetting such test pages with different fonts are shown in Figures 8–11, except that each font sample was usually typeset at least twice with LuaLaTeX and XeLaTeX.

In some cases, we also tested an additional version with ConTeXt MkIV, but unfortunately we had to use a modified version of our test files in such cases.

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2}.$$

Figure 4: Comparison of math typesetting from XeLaTeX (red) and LuaLaTeX (blue) using Cambria Math. (The example shows the definition of the Laplace operator in vector analysis.)

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}.$$

Figure 5: Comparison of math typesetting from XeLaTeX (red) and LuaLaTeX (blue) using Cambria Math. (The example shows the Einstein field equation from general relativity.)

Analyzing large-scale effects

Comparing the different versions for the same font typeset with different engines or macro packages may reveal significant differences at various scales.

If there are any unexpected large-scale effects, such as using different sizes of delimiters or operators, it is usually easy to spot them simply by visual inspection. In most cases, such obvious differences will turn out to be unintentional and tend to indicate problems or bugs, such as those discussed earlier in this paper.

Analyzing small-scale effects

Once we have applied all the necessary workarounds and bug fixes to eliminate the unexpected large-scale effects, only small-scale effects should remain, affecting tiny micro-typographic details, which may be hard to see without closer inspection.

In order to the study the small-scale effects in more detail, we created another set of more sophisticated test files using PDF overlays between different versions of the same document using different colors.

These overlays were generated with PDFLaTeX using the pdfpages package as follows:

```
\documentclass[a4paper]{article}
\usepackage{pdfpages}

\begin{document}
\includepdfmerge[nup=2x1,noautoscale=true,
delta=-21cm 0] % width of A4 paper
{xelatex-test.pdf,1,lualatex-test.pdf,1,
...
xelatex-test.pdf,n,lualatex-test.pdf,n}
\end{document}
```

This setup will put each page of the LuaLaTeX test file on top of the corresponding page of the XeLaTeX test file. For improved visibility, colors should be chosen in such a way that the darker colors (such as black or blue) will appear on top of the brighter colors (such as red). In our example illustrations we have usually used red for XeLaTeX overlaid by blue for LuaLaTeX.

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}.$$

Figure 6: Comparison of math typesetting from XeLaTeX (red) and LuaLaTeX (blue) after applying a workaround for XeLaTeX to circumvent inconsistent placement of superscripts.

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}.$$

Figure 7: Comparison of math typesetting from XeLaTeX (red) and LuaLaTeX (blue) after applying a workaround for XeLaTeX to circumvent inconsistent placement of superscripts.

Analyzing the results of overlays

Once we have generated overlays of the results from typesetting the same equations with different engines, it is easy to detect if any differences occur. However, it is far from easy to understand how these differences come about and what their implications might be.

In our first series of tests, we originally noticed some very significant differences in vertical spacing around fraction bars. Once we detected the problem of XeTeX incorrectly using the preloaded set of font parameters and applied a workaround, most differences in vertical spacing disappeared and only few remained.

In our second series of tests, only relatively few differences remained. Moreover, the remaining effects only appeared for some fonts and not for others. While there were hardly any effects on vertical spacing for XITS Math, there were some notable differences in the placement of scripts for Asana Math or Cambria Math, as illustrated in Figures 4–5.

Upon closer inspection, we eventually found that the differences only affected some letters within an equation, but not all of them. There were no differences on letters without ascenders or descenders (as in x_0 or x^2), while there were differences for superscripts on letters with ascenders (as in ∂^2) and also for subscripts on letters with descenders (as in μ_0). The cause of this problem isn't clear yet, but it most likely indicates an unresolved engine problem in XeTeX.

In our third series of tests, the remaining effects on vertical spacing could be eliminated completely after we applied a workaround for the placement of scripts, and only some effects on horizontal spacing remained, as illustrated in Figures 6–7.

The remaining effects on horizontal spacing are most likely related to different interpretations of OpenType glyph metrics in different TeX engines (such as italic corrections and math kerning [18]), which certainly will have an effect on the quality of math typesetting, but only on a very small scale.

Summary and Conclusions

In this paper, we have reported our experiences, findings and observations from testing OpenType math support in TeX Live 2010 with different TeX engines, macro packages and fonts.

When we set out, we expected to gain some insights how the quality of math typesetting was affected by the use of additional font parameters in a more sophisticated implementation of OpenType math support.

In the end, however, it turned out that most of the differences were actually caused by unresolved bugs in both macro packages and TeX engines, while the use of additional math font parameters appears to be largely irrelevant for our selection of test cases.

It was only by direct comparison with LuaTeX that some long-standing bugs or inconsistencies in XeTeX engine and the unicode-math package could be found. Without a suitable baseline reference, it is obviously hard to tell if the spacing of math is exactly right or just slightly wrong, so it is not surprising that minor inconsistencies went unnoticed for a long time.

Once we applied workarounds or fixes for the problems we discovered, the remaining differences between different TeX engines turned out to be much smaller than expected and only affected the horizontal spacing, but no longer the vertical spacing.

Given the scale of the remaining effects, our studies regarding the quality of math typesetting in different TeX engines remain inconclusive for now. Both engines can produce very similar results, but XeTeX will do so only after applying a number of fixes and workarounds to arrive at what LuaTeX will do by default.

Acknowledgements

The author would like to thank the developers involved in math-related TeX engines, macro packages and fonts for their assistance and feedback during the testing of OpenType math font support in TeX Live 2010.

In particular, Will Robertson (unicode-math), Khaled Hosny (luaotfload), Taco Hoekwater (LuaTeX), Hans Hagen (ConTeXt), Jonathan Kew (XeTeX), Karl Berry and Peter Breitenlohner (TeX Live 64-bit Linux binaries) contributed to our testing and problem solving efforts in one way or another.

Fortunately, we were able to discover and eliminate a number of bugs before the deadline of TeX Live 2010 pretest. Unfortunately, not all known issues could be resolved in time, so some problems remain to be fixed in future releases. While such fixes for macro packages and fonts can be issued through the TeX Live update mechanism at any time, fixes for TeX engines may be delayed until next year's TeX Live release.

Cambria Math Example

Vector calculus:

$$\nabla\phi(\mathbf{r}) = \frac{\partial\phi}{\partial x}\mathbf{e}_x + \frac{\partial\phi}{\partial y}\mathbf{e}_y + \frac{\partial\phi}{\partial z}\mathbf{e}_z,$$

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}.$$

Maxwell equations (differential form):

$$\operatorname{div}\varepsilon_0\mathbf{E} = \lambda, \quad \operatorname{div}\mathbf{B} = 0,$$

$$\operatorname{rot}\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \quad \operatorname{rot}\frac{\mathbf{B}}{\mu_0} = \mathbf{j} + \frac{\partial\varepsilon_0\mathbf{E}}{\partial t}.$$

Maxwell equations (integral form):

$$\int_F \varepsilon_0\mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0,$$

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_F \mathbf{B} \cdot d\mathbf{f},$$

$$\oint_C \frac{\mathbf{B}}{\mu_0} \cdot d\mathbf{l} = \int_F \left(\mathbf{j} + \frac{\partial\varepsilon_0\mathbf{E}}{\partial t} \right) \cdot d\mathbf{f}.$$

Electromagnetic wave equations:

$$\Delta\mathbf{E} - \frac{1}{c^2} \frac{\partial^2\mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0} \nabla\lambda + \mu_0 \frac{\partial\mathbf{j}}{\partial t},$$

$$\Delta\mathbf{B} - \frac{1}{c^2} \frac{\partial^2\mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot}\mathbf{j}.$$

Energy-mass equation (special relativity):

$$E = \frac{m_0 c^2}{\sqrt{1 - v^2/c^2}}.$$

Einstein field equation (general relativity):

$$R^{\mu\nu} - \frac{1}{2} R g^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2} M^{\mu\nu}.$$

Schrödinger equation (quantum mechanics):

$$i\hbar \frac{\partial\psi}{\partial t} = \hat{H}\psi = \frac{1}{2m} \left(\frac{\hbar}{i} \nabla - q\mathbf{A} \right)^2 \psi + q\phi\psi.$$

Dirac equation (relativistic quantum mechanics):

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0.$$

Figure 8: Sampling of equations typeset with LuaLaTeX using Cambria and Cambria Math.

Asana Math Example

Vector calculus:

$$\nabla\phi(\mathbf{r}) = \frac{\partial\phi}{\partial x}\mathbf{e}_x + \frac{\partial\phi}{\partial y}\mathbf{e}_y + \frac{\partial\phi}{\partial z}\mathbf{e}_z,$$

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}.$$

Maxwell equations (differential form):

$$\operatorname{div}\varepsilon_0\mathbf{E} = \lambda, \quad \operatorname{div}\mathbf{B} = 0,$$

$$\operatorname{rot}\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \quad \operatorname{rot}\frac{\mathbf{B}}{\mu_0} = \mathbf{j} + \frac{\partial\varepsilon_0\mathbf{E}}{\partial t}.$$

Maxwell equations (integral form):

$$\int_F \varepsilon_0\mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0,$$

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_F \mathbf{B} \cdot d\mathbf{f},$$

$$\oint_C \frac{\mathbf{B}}{\mu_0} \cdot d\mathbf{l} = \int_F \left(\mathbf{j} + \frac{\partial\varepsilon_0\mathbf{E}}{\partial t} \right) \cdot d\mathbf{f}.$$

Electromagnetic wave equations:

$$\Delta\mathbf{E} - \frac{1}{c^2} \frac{\partial^2\mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0} \nabla\lambda + \mu_0 \frac{\partial\mathbf{j}}{\partial t},$$

$$\Delta\mathbf{B} - \frac{1}{c^2} \frac{\partial^2\mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot}\mathbf{j}.$$

Energy-mass equation (special relativity):

$$E = \frac{m_0 c^2}{\sqrt{1 - v^2/c^2}}.$$

Einstein field equation (general relativity):

$$R^{\mu\nu} - \frac{1}{2} R g^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2} M^{\mu\nu}.$$

Schrödinger equation (quantum mechanics):

$$i\hbar \frac{\partial\psi}{\partial t} = \hat{H}\psi = \frac{1}{2m} \left(\frac{\hbar}{i} \nabla - q\mathbf{A} \right)^2 \psi + q\phi\psi.$$

Dirac equation (relativistic quantum mechanics):

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0.$$

Figure 9: Sampling of equations typeset with LuaLaTeX using TeX Gyre Pagella and Asana Math.

XITS Math Example

Vector calculus:

$$\nabla\phi(\mathbf{r}) = \frac{\partial\phi}{\partial x}\mathbf{e}_x + \frac{\partial\phi}{\partial y}\mathbf{e}_y + \frac{\partial\phi}{\partial z}\mathbf{e}_z,$$

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}.$$

Maxwell equations (differential form):

$$\operatorname{div}\epsilon_0\mathbf{E} = \lambda, \quad \operatorname{div}\mathbf{B} = 0,$$

$$\operatorname{rot}\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \quad \operatorname{rot}\frac{\mathbf{B}}{\mu_0} = \mathbf{j} + \frac{\partial\epsilon_0\mathbf{E}}{\partial t}.$$

Maxwell equations (integral form):

$$\int_F \epsilon_0\mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0,$$

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_F \mathbf{B} \cdot d\mathbf{f},$$

$$\oint_C \frac{\mathbf{B}}{\mu_0} \cdot d\mathbf{l} = \int_F \left(\mathbf{j} + \frac{\partial\epsilon_0\mathbf{E}}{\partial t} \right) \cdot d\mathbf{f}.$$

Electromagnetic wave equations:

$$\Delta\mathbf{E} - \frac{1}{c^2} \frac{\partial^2\mathbf{E}}{\partial t^2} = \frac{1}{\epsilon_0} \nabla\lambda + \mu_0 \frac{\partial\mathbf{j}}{\partial t},$$

$$\Delta\mathbf{B} - \frac{1}{c^2} \frac{\partial^2\mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot}\mathbf{j}.$$

Energy-mass equation (special relativity):

$$E = \frac{m_0 c^2}{\sqrt{1 - v^2/c^2}}.$$

Einstein field equation (general relativity):

$$R^{\mu\nu} - \frac{1}{2} R g^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2} M^{\mu\nu}.$$

Schrödinger equation (quantum mechanics):

$$i\hbar \frac{\partial\psi}{\partial t} = \hat{H}\psi = \frac{1}{2m} \left(\frac{\hbar}{i} \nabla - q\mathbf{A} \right)^2 \psi + q\phi\psi.$$

Dirac equation (relativistic quantum mechanics):

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0.$$

Figure 10: Sampling of equations typeset with LuaLaTeX using XITS and XITS Math.

Neo Euler Example

Vector calculus:

$$\nabla\phi(\mathbf{r}) = \frac{\partial\phi}{\partial x}\mathbf{e}_x + \frac{\partial\phi}{\partial y}\mathbf{e}_y + \frac{\partial\phi}{\partial z}\mathbf{e}_z,$$

$$\Delta\phi(\mathbf{r}) = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}.$$

Maxwell equations (differential form):

$$\operatorname{div}\epsilon_0\mathbf{E} = \lambda, \quad \operatorname{div}\mathbf{B} = 0,$$

$$\operatorname{rot}\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}, \quad \operatorname{rot}\frac{\mathbf{B}}{\mu_0} = \mathbf{j} + \frac{\partial\epsilon_0\mathbf{E}}{\partial t}.$$

Maxwell equations (integral form):

$$\int_F \epsilon_0\mathbf{E} \cdot d\mathbf{f} = \int_V \lambda dV, \quad \int_F \mathbf{B} \cdot d\mathbf{f} = 0,$$

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_F \mathbf{B} \cdot d\mathbf{f},$$

$$\oint_C \frac{\mathbf{B}}{\mu_0} \cdot d\mathbf{l} = \int_F \left(\mathbf{j} + \frac{\partial\epsilon_0\mathbf{E}}{\partial t} \right) \cdot d\mathbf{f}.$$

Electromagnetic wave equations:

$$\Delta\mathbf{E} - \frac{1}{c^2} \frac{\partial^2\mathbf{E}}{\partial t^2} = \frac{1}{\epsilon_0} \nabla\lambda + \mu_0 \frac{\partial\mathbf{j}}{\partial t},$$

$$\Delta\mathbf{B} - \frac{1}{c^2} \frac{\partial^2\mathbf{B}}{\partial t^2} = -\mu_0 \operatorname{rot}\mathbf{j}.$$

Energy-mass equation (special relativity):

$$E = \frac{m_0 c^2}{\sqrt{1 - v^2/c^2}}.$$

Einstein field equation (general relativity):

$$R^{\mu\nu} - \frac{1}{2} R g^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2} M^{\mu\nu}.$$

Schrödinger equation (quantum mechanics):

$$i\hbar \frac{\partial\psi}{\partial t} = \hat{H}\psi = \frac{1}{2m} \left(\frac{\hbar}{i} \nabla - q\mathbf{A} \right)^2 \psi + q\phi\psi.$$

Dirac equation (relativistic quantum mechanics):

$$\gamma^\alpha \left(\frac{\hbar}{i} \partial_\alpha - qA_\alpha \right) \psi + m_0 c \psi = 0.$$

Figure 11: Sampling of equations typeset with LuaLaTeX using T_{EX} Gyre Pagella and Neo Euler.

References

- [1] T_EX Users Group: Testing T_EX Live before release. <http://tug.org/texlive/pretest>
- [2] Barbara Beeton, Asmus Freytag, Murray Sargent III: Unicode Support for Mathematics. Unicode Technical Report UTR#25. 2001. <http://www.unicode.org/reports/tr25/>
- [3] Barbara Beeton: Unicode and math, a combination whose time has come. *TUGboat*, 21(3):174–185, 2000. Proceedings of TUG 2000, Oxford, UK. <http://www.tug.org/TUGboat/tb21-3/tb68beet.pdf>
- [4] Barbara Beeton: The STIX Project – From Unicode to fonts. *TUGboat*, 28(3):299–304, 2007. Proceedings of TUG 2007, San Diego, CA, USA. <http://www.tug.org/TUGboat/tb28-3/tb90beet.pdf>
- [5] Ulrik Vieth: Math Typesetting in T_EX: The Good, The Bad, The Ugly. *MAPS*, 26:207–216, 2001. Proceedings of EuroT_EX 2001, Kerkrade, Netherlands. <http://www.ntg.nl/maps/26/27.pdf>
- [6] OpenType Specification, Version 1.6. <http://www.microsoft.com/typography/otspec/>
- [7] Murray Sargent III: Math in Office Blog. <http://blogs.msdn.com/murrays/default.aspx>
- [8] John Hudson, Ross Mills: Mathematical Typesetting: Mathematical and scientific typesetting solutions. Promotional Booklet, Microsoft, 2006.
- [9] Daniel Rhatigan: Three typefaces for mathematics. Dissertation for the MA in typeface design, 2007. http://www.typeculture.com/academic_resource/articles_essays/pdfs/tc_article_47.pdf
- [10] Murray Sargent III: Unicode Nearly Plain Text Encodings of Mathematics. Unicode Technical Note UTN#28, 2006. <http://www.unicode.org/notes/tn28/>
- [11] George Williams: FontForge: Math typesetting information. <http://fontforge.sourceforge.net/math.html>
- [12] Ulrik Vieth: Do we need a ‘Cork’ math font encoding? *TUGboat*, 29(3):426–434, 2008. Proceedings of TUG 2008, Cork, Ireland. <http://www.tug.org/TUGboat/tb29-3/tb93vieth.pdf>
- [13] Ulrik Vieth: OpenType Math Illuminated. Reprinted in *TUGboat*, 30(1):22–31, 2009. Proceedings of BachoT_EX 2009, Bachotek, Poland. <http://www.tug.org/TUGboat/tb30-1/tb94vieth.pdf>
- [14] Boguslaw Jackowski: Appendix G Illuminated. *TUGboat*, 27(1):83–90, 2006. Proceedings of EuroT_EX 2006, Debrecen, Hungary. <http://www.tug.org/TUGboat/tb27-1/tb86jackowski.pdf>
- [15] Ulrik Vieth: Understanding the aesthetics of math typesetting. *Biuletyn GUST*, 5–12, 2008. Proceedings of BachoT_EX 2008, Bachotek, Poland. <http://www.gust.org.pl/projects/e-foundry/math-support/vieth2008.pdf>
- [16] Jonathan Kew: XeT_EX Live. *TUGboat*, 29(1):151–156, 2008. Proceedings of BachoT_EX 2007, Bachotek, Poland. <http://www.tug.org/TUGboat/tb29-1/tb91kew.pdf>
- [17] Taco Hoekwater: LuaT_EX Reference Manual. <http://www.luaotex.org/svn/trunk/manual/Luatexref-t.pdf>
- [18] Taco Hoekwater: Math in LuaT_EX 0.40. *MAPS*, 38:22–31, 2009. <http://www.ntg.nl/maps/38/04.pdf>
- [19] Hans Hagen: Unicode Math in ConT_EXt. *MAPS*, 38:32–46, 2009. <http://www.ntg.nl/maps/38/05.pdf>
- [20] Will Robertson: Advanced font features with XeT_EX: The fontspec package. *TUGboat*, 26(3):215–223, 2005. <http://www.tug.org/TUGboat/tb26-3/tb84robertson.pdf>
- [21] Will Robertson: The fontspec macro package. <http://www.ctan.org/pkg/fontspec> <http://github.com/wspr/fontspec>
- [22] Will Robertson: The unicode-math macro package. <http://www.ctan.org/pkg/unicode-math> <http://github.com/wspr/unicode-math>
- [23] Aditya Mahajan: Integrating Unicode and OpenType math in ConT_EXt. *TUGboat*, 30(2):243–246, 2009. Proceedings of TUG 2009, Notre Dame, IN, USA. <https://www.tug.org/members/TUGboat/tb30-2/tb95mahajan-cmath.pdf>
- [24] Khaled Hosny et al.: The luaotfload macro package. <http://www.ctan.org/pkg/luaotfload> <http://github.com/khaledhosny/luaotfload>
- [25] Will Robertson: Unicode mathematics in LaT_EX: advantages and challenges. To appear in *TUGboat*, 31(2):???–???, 2010. Proceedings of TUG 2010, San Francisco, CA, USA. <https://www.tug.org/members/TUGboat/tb31-2/tb98robertson.pdf>
- [26] Ulrik Vieth: Experiences typesetting mathematical physics. *MAPS*, 39:166–178, 2009. Proceedings of EuroT_EX 2009, Delft, Netherlands. <https://www.tug.org/members/TUGboat/tb30-3/tb96vieth.pdf>
- [27] Apostolos Syropoulos: Asana Math Font. <http://www.ctan.org/pkg/asana-math>
- [28] Khaled Hosny: XITS Fonts. <http://www.ctan.org/pkg/xits> <http://github.com/khaledhosny/xits-math>
- [29] STIX Consortium: STIX Fonts. <http://www.stixfonts.org/> <http://www.ctan.org/pkg/stix>
- [30] Khaled Hosny: Neo Euler Font. <http://github.com/khaledhosny/euler-otf>
- [31] Hans Hagen, Taco Hoekwater, Volker RW Schaa: Reshaping Euler: A collaboration with Hermann Zapf. *TUGboat*, 29(3):283–287, 2008. <http://www.tug.org/TUGboat/tb29-2/tb92hagen-euler.pdf>

Ulrik Vieth
Vaihinger Straße 69
70567 Stuttgart
Germany
ulrik dot vieth (at) arcor dot de

Abstract: T_EXLive 2010 will contain LuaT_EX 0.60. This article gives an overview of the changes between this version and the version on last year's T_EXLive. Highlights of this release: cweb code base, dynamic loading of lua modules, various font subsystem improvements including support for Apple .dfont, font collection files, braced input file names, extended pdf Lua table, and access to the line breaking algorithm from Lua code.

Key words: Lua, LuaT_EX, verze 0.60

Abstrakt: Článek představuje změny a novinky ve verzi LuaT_EXu 0.60.

Klíčová slova: programovací jazyk Lua, LuaT_EX, verze 0.60

References

- [1] LuaT_EX Available at URL: <http://www.luatex.org/>
- [2] The Programming Language Lua. Home page. Available at URL: <http://www.lua.org/>

taco (at) elvenkind (dot) com
Elvenkind BV, Spuiboulevard 269, 3311 GP Dordrecht, The Netherlands

General changes

Some of the changes can be organised into sections, but not all. So first, here are the changes that are more or less standalone.

- Many of the source files have been converted to cweb. Early versions of Lua_{TeX} were based on Pascal web, but by 0.40 all code has been hand-converted to C. The literate programming comments were kept, and the relevant sources have now been converted back to cweb, reinstating the literate documentation.

This change does not make Lua_{TeX} a literate program in the traditional sense because the typical C source code layout with pairs of header & implementation files has been kept and no code reshuffling takes place. But it does mean that it is now much easier to keep the source documentation up to date, and it is possible to create nicely typeset program listings with indices.

- There are now source repository revision numbers in the banner again, which is a useful thing to have while tracking down bugs. For example, the Lua_{TeX} binary being used to write this article starts up with:

```
This is LuaTeX, Version beta-0.60.1-2010042817 (rev 3659)
```

- The horizontal nodes that are added during line breaking now inherit the attributes from the nodes inside the created line.

Previously, these nodes (`\leftskip` and `\rightskip` in particular) inherited the attributes in effect at the end of the (partial) paragraph because that is where line breaking takes place.

- All Lua errors now report file and line numbers to aid in debugging, even if the error happens inside a callback.
- Lua_{TeX} can now use the embedded kpathsea library to find Lua `require()` files, and will do so by default if the kpathsea library is enabled by the format (as is the case in plain Lua_{TeX} and the various Lua^L_{TeX} formats).
- The print precision for small numbers in Lua code (the return value of `tostring()`) has been improved.
- Of course there were lots of code cleanups and improvements to the reference manual.

Embedded libraries and other third-party inclusions

The following are changes to third-party code that for the most part should not need much explanation.

- MetaPost is now at version 1.211.
- Libpng is now at version 1.2.40.
- New Sync_{TeX} code is imported from _{TeX} Live.
- The Lua source file from the `luamd5` library (which provides the `md5_hexsum` function) is now embedded in the executable. In older versions of Lua_{TeX}, this file was missing completely.
- The Lua co-routine patch (`coco`) is now disabled on `powerpc-linux` because it caused crashes on that platform due to a bad upstream implementation.

Dynamic loading of lua modules

Lua_{TeX} now has support for dynamic loading of external compiled Lua libraries.

As with other `require()` files, Lua_{TeX} can and will use kpathsea if the format allows it to do so. For this purpose, kpathsea has been extended with a new file type: `clua`. The associated `texmf.cnf` variable is by default defined like this:

```
CLUAINPUTS=.:$SELFAUTOLOC/lib/{"$progname,$engine,}/lua//
```

which means that if your LuaTeX binary lives in

```
/opt/tex/texmf-linux-64/bin/
```

then your compiled Lua modules should go into the local directory, or in a tree below

```
/opt/tex/texmf-linux-64/bin/lib/lua
```

Be warned that not all available Lua modules will work. LuaTeX is a command line program, and on some platforms that makes it nearly impossible to use GUI-based extensions.

Font related

Lots of small changes have taken place in the font processing.

- The backend message

```
cannot open Type 1 font file for reading
```

now reports the name of the Type1 font file it was looking for.

- It is no longer possible for fonts from included pdf files to be replaced by / merged with the document fonts of the enveloping pdf.
- Support for Type3 .pgc files has been removed. This is just for the .pgc format invented by Hàn Thê Thành, bitmapped pk files still work.
- For TrueType font collections (.ttc files), the used subfont name and its index id are printed to the terminal, and if the backend cannot find the font in the collection, the run is aborted.
- It is now possible to use Apple .dfont font collection files.

Unfortunately, in Snow Leopard (a.k.a. MacOSX 10.6) Apple switched to a .ttc format that is not quite compatible with the Microsoft version of .ttc. As a result, the system fonts from Snow Leopard cannot be used in LuaTeX 0.60.

- The loading speed of large fonts via the fontloader library, and the inclusion speed for sub-setting in the backend have both been improved.
- There are two new MathConstants entries added. Suppose the Lua math font loading code produces a Lua table named `f`. In that table, you can set

```
f.MathConstants.FractionDelimiterSize  
f.Mathconstants.FractionDelimiterDisplayStyleSize
```

These new fields allows proper setting of the size parameters for LuaTeX's `...withdelims` math primitives, for which there is no ready replacement in the OpenType MATH table.

- Artificially slanted or extended fonts now work via the pdf text matrix so that this now also works for non-Type1 fonts. In other words: the Lua `f.slant` and `f.extend` font keys are now obeyed in all cases.
- There is another new allowed key: `f.psname`. When set, this value should be the original PostScript font name of the font. In the pdf generation backend, fonts inside .dfont and .ttc collections are fetched from the archive using this field, so in those cases the key is required.

- A related change is made to the font name discovery used by the backend for storage into the pdf file structure: now it tries `f.psnam` first, as that is much less likely to contain spaces than `f.fontname` (which is the field that 0.40 used). If there is no `f.psnam`, it falls back to the old behaviour.
- Finally, Lua-loaded fonts now support a `f.nomath` key to speed up loading the Lua table in the normal case of fonts that do not provide OpenType MATH data.

'T_EX'-side extensions and changes

LuaT_EX is not actually T_EX even though it uses an input language that is very similar, hence the quotes in this section's title. Some of the following items are new LuaT_EX extensions, others are adjustments to pre-existing pdfT_EX or Aleph functionality.

- The primitives `\input` and `\openin` now accept braced file names, removing the need for double quote escapes in case of files with spaces in their name.
- The `\endlinechar` can now be set to any value between 0 and 127.
- The new primitives `\aligntab` and `\alignmark` are aliases for the use single characters with the category codes of `&` and `#` in alignments.
- `\latelua` is now allowed inside leaders. To be used with care, because the Lua code will be executed once for each generated leader item.
- The new primitive `\gleaders` provides 'globally aligned' leaders. These leaders are aligned on one side of the main output box instead of to the side of the immediately enclosing box.
- From now on LuaT_EX handles only 4 direction specifiers:
 - TLT (latin),
 - TRT (arabic),
 - RTT (cjk), and
 - LTL (mongolian).
 Other direction specifiers generate an error.
- The `\pdfcompresslevel` is now effectively fixed as soon as any output to the pdf file has occurred.
- `\pdfobj` has gained an extra optional keyword: `uncompressed`. This forces the object to be written to the pdf in plain text, which is needed for certain objects containing metadata.
- Two new token lists are provided: `\pdfxformresources` and `\pdfxformattr`, as an alternative to `\pdfxform` keywords.
- The new syntax

```
\pdfrefxform [width <dimen>] [height <dimen>] [depth <dimen>] <formref>
```

scales a single form object; using similar principle as with `\pdfximage`: depth alone doesn't scale, it shifts vertically.

- Similarly,

```
\pdfrefximage [width <dimen>] [height <dimen>] [depth <dimen>] <imageref>
```

overrules settings from `\pdfximage` for this image only.

- The following obsolete pdfT_EX primitives have been removed:
 - `\pdfoptionalwaysusepdfpagebox`
 - `\pdfoptionpdfinclusionerrorlevel`
 - `\pdfforcepagebox`
 - `\pdfmovechars`

These were already deprecated in pdfTeX itself.

Lua table extensions

In most of the Lua tables that LuaTeX provides, only small changes have taken place, so they do not deserve their own subsections.

- There is a new callback: `process_output_buffer`, for post-processing of `\write` text to a file.
- The callbacks `hpack_filter`, `vpack_filter` and `pre_output_filter` pass on an extra string argument for the current direction.
- `fontloader.open()` previously cleared some of the font name strings during load that it should not do.
- The new function `font.id("tenrm")` returns the internal id number for that font. It takes a bare control sequence name as argument.
- The `os.name` variable now knows about cygwin and kfreebsd.
- `lfs.readlink("file")` returns the content of a symbolic link (Unix only). This extension is meant for use in texlua scripts.
- `lfs.shortname("file")` returns the short (FAT) name of a file (Windows only). This extension is meant for use in texlua scripts.
- `kpse.version()` returns the kpathsea version string.
- `kpse.lookup({...})` offers a search interface similar to the `kpsewhich` program, an example call looks like this:

```
kpse.set_program_name('luatex')
print(kpse.lookup ('plain.tex',
                  { ["format"] = "tex",
                    ["all"] = true,
                    ["must-exist"] = true })))
```

The 'node' table

In the verbatim code below, `n` stands for a userdata node object.

- `node.vpack(n)` packs a list into a vlist node, like `\vbox`.
- `node.protrusion_skippable(n)` returns true if this node can be skipped for the purpose of protrusion discovery.

This is useful if you want to (re)calculate protrusion in pure Lua.

- `node.dimensions(n)` returns the natural width, height and depth of a (horizontal) node list.
- `node.tail(n)` returns the tail node of a node list.
- Each glyph node now has three new virtual read-only fields: `width`, `height`, and `depth`. The values are the number of scaled points.
- `glue_spec` nodes now have an extra boolean read-only field: `writable`.

Some glue specifications can be altered directly, but certain key glue specifications are shared among many nodes. Altering the values of those is prohibited because it would have unpredictable side-effects. For those cases, a copy must be made and assigned to the parent node.

- `hlist` nodes now have a subtype to distinguish between `hlists` generated by the paragraph breaking, explicit `\hbox` commands, and other sources.
- `node.copy_list(n)` now allows a second argument. This argument can be used to copy only part of a node list.
- `node.hpack(n)` now accepts `cal_expand_ratio` and `subst_ex_font` modifiers.

This feature helps the implementation of font expansion in a pure Lua paragraph breaking code.

- `node.hpack(n)` and `node.vpack(n)` now also return the ‘badness’ of the created box, and accept an optional direction argument.

The ‘pdf’ table

- The new functions `pdf.mapfile("...")` and `pdf.mapline("...")` are aliases for the corresponding pdf \TeX primitives.
- `pdf.registerannot()` reserves a pdf object number and returns it.
- The functions `pdf.obj(...)`, `pdf.immediateobj(...)`, and `pdf.reserveobj(...)` are similar to the corresponding pdf \TeX primitives. Full syntax details can be read in the Lua \TeX reference manual.
- New read-write string keys:
 - `pdf.catalog` string goes into the Catalog dictionary.
 - `pdf.info` string goes into the Info dictionary.
 - `pdf.names` string goes into the Names dictionary, referenced by the Catalog object.
 - `pdf.trailer` string goes into the Trailer dictionary.
 - `pdf.pageattributes` string goes into the Page dictionary.
 - `pdf.pageresources` string goes into the Resources dictionary referenced by the Page object.
 - `pdf.pagesattributes` string goes into the Pages dictionary.

The ‘tex’ table

Finally, there are some extensions to the `tex` table that are worth mentioning.

- `tex.badness(f,s)` interfaces to the ‘badness’ internal function.
 - Accidentally, this disables access to the `\badness` internal parameter, this will be corrected in a future Lua \TeX version.
- `tex.sp("in")` converts Lua-style string units to scaled points.
- `tex.tprint({...},{...})` is like a sequence of `tex.sprint(...)` calls.
- `tex.shipout(n)` ships out a constructed box.
- `tex.nest[]` and `tex.nest.ptr` together allow read-write access to the semantic nest (mode nesting).

For example, this prints the equivalent of `\prevdepth` at the current mode nesting level.

```
print (tex.nest[tex.nest.ptr].prevdepth)
```

`tex.nest.ptr` is the current level, and lower numbers are enclosing modes.

Each of the items in the `tex.nest` array represents a mode nesting level and has a set of virtual keys that be accessed both for reading and writing, but you cannot change the actual `tex.nest` array itself. The possible keys are listed in the Lua \TeX reference manual.

- `tex.linebreak(n, {...})` supports running the paragraph breaker from pure Lua. The second argument specifies a (potentially large) table of line breaking parameters: the parameters that are not passed explicitly are taken from the current typesetter state.

The exact keys in the table are documented in the reference manual, but here is a simple yet complete example of how to run line breaking on the content of `\box0`:

```

\setbox0=\hbox to \hsize{\input knuth }
\startluacode
local n = node.copy_list(tex.box[0].list)
local t = node.tail(n)
local final = node.new(node.id('glue'))
final.spec = node.new(node.id('glue_spec'))
final.spec.stretch_order = 2
final.spec.stretch = 1
node.insert_after(n,t, final)
local m = tex.linebreak(n,
    { hangafter = 2, hangindent = tex.sp("2em")})
local q = node.vpack(m)
node.write(q)
\stoptluacode

```

The result is this:

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large-scale user; the designer should also write the first user manual. The separation of any of these four components would have hurt \TeX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important. But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

Summary

All in all, there are not too many incompatible changes compared to Lua \TeX 0.40, and the Lua \TeX project is progressing nicely.

Lua \TeX beta 0.70 will be released in the autumn of 2010. Our current plans for that release are: access to the actual pdf structures of included pdf images; a partial redesign of the mixed direction model; even more access to the Lua \TeX internals from Lua; and probably some more ...

LuaT_EX 0.63 Short Reference

LuaT_EX 0.63 Stručný průvodce

TACO HOEKWATER

Abstract: This manual gives a brief description of functions collected mainly in the callback, font, font loader, image, kpathsea, language, lua, metapost, node, pdf, status, typesetting, IO, texconfig and token tables provided by LuaT_EX 0.63.

Key words: Lua, LuaT_EX, revision 0, version 63, tables

Abstrakt: V manuálu je uveden stručný popis funkcí, zahrnutých zejména ve zpětných voláních, v řadě tabulek, které poskytuje LuaT_EX 0.63.

Klíčová slova: programovací jazyk Lua, LuaT_EX, verze 0.63, tabulky

*taco (at) elvenkind (dot) com
Elvenkind BV, Spuiboulevard 269, 3311 GP Dordrecht, The Netherlands*

Tokenisation changes callbacks

`string = process_input_buffer(string)`
 Modify the encoding of the input buffer.
`string = process_output_buffer(string)` Modify the encoding of the output buffer.
`table = token_filter()` Override the tokenization process. Return value is a token or an array of tokens

Node list callbacks

`buildpage_filter(string)` Process objects as they are added to the main vertical list. The string argument gives some context.

`buildpage_filter` context information:

value	explanation
alignment	a (partial) alignment is being added
after_output	an output routine has just finished
box	a typeset box is being added
new_graf	the beginning of a new paragraph
vmode_par	\par was found in vertical mode
hmode_par	\par was found in horizontal mode
insert	an insert is added
penalty	a penalty (in vertical mode)
before_display	immediately before a display starts
after_display	a display is finished
end	LuaTeX is terminating (it's all over)

`node = pre_linebreak_filter(node, string)`
 Alter a node list before linebreaking takes place. The string argument gives some context.

`pre_linebreak_filter` context information:

value	explanation
<empty>	main vertical list
hbox	\hbox in horizontal mode
adjusted_hbox	\hbox in vertical mode
vbox	\vbox
vtop	\vtop
align	\halign or \valign
disc	discretionaries
insert	packaging an insert
vcenter	\vcenter
local_box	\localleftbox or \localrightbox
split_off	top of a \vsplit
split_keep	remainder of a \vsplit
align_set	alignment cell
fin_row	alignment row

`node = linebreak_filter(node, boolean)`
 Override the linebreaking algorithm. The boolean is true if this is a pre-display break.

`node = post_linebreak_filter(node, string)` Alter a node list after line-breaking has taken place. The string argument gives some context.

`node = hpack_filter(node, string, number, string, string)` Alter a node list before horizontal packing takes place. The first string gives some context, the number is the desired size, the second string is either "exact" or "additional" (modifies the first string), the third string is the desired direction

`node = vpack_filter(node, string, number, string, number, string)` Alter a node list before vertical packing takes place. The second number is the desired max depth. See `hpack_filter` for the arguments.

`node = pre_output_filter(node, string, number, string, number, string)`
 Alter a node list before boxing to \outputbox takes place. See `vpack_filter` for the arguments.

`hyphenate(node, node)` Apply hyphenation to a node list.

`ligaturing(node, node)` Apply ligaturing to a node list.

`kerning(node, node)` Apply kerning to a node list.

`node = mlist_to_hlist(node, string, boolean)` Convert a math node list into a horizontal node list.

Font definition callback

`metrics = define_font(string, number)` Define a font from within lua code. The arguments are the user-supplied information, with negative numbers indicating scaled, positive numbers at

Event callbacks

`pre_dump()` Run actions just before format dumping takes place.
`stop_run()` Run actions just before the end of the typesetting run.
`start_run()` Run actions at the start of the typesetting run.
`start_page_number()` Run actions at the start of typeset page number message reporting.
`stop_page_number()` Run actions at the end of typeset page number message reporting.
`show_error_hook()` Run action at error reporting time.
`finish_pdffile()` Run actions just before the PDF closing takes place.

Font table

`metrics = font.read_tfm(string, number)` Parse a font metrics file, at the size indicated by the number.

`metrics = font.read_vf(string, number)`

Parse a virtual font metrics file, at the size indicated by the number.

`metrics = font.getfont(number)` Fetch an internal font id as a lua table.

`font.setfont(number, metrics)` Set an internal font id from a lua table.

`boolean = font.frozen(number)` True if the font is frozen and can no longer be altered.

`number = font.define(metrics)` Process a font metrics table and stores it in the internal font table, returning its internal id.

`number = font.nextid()` Return the next free font id number.

`number = font.id(string)` Return the font id of the font accessed by the cfname given.

`[number] = font.current([number])` Get or set the currently active font

`number = font.max()` Return the highest used font id at this moment.

`number, metrics = font.each()` Iterate over all the defined fonts.

Font loader table

`table = fontloader.info(string)` Get various information fields from an font file.

`fontloader.info` returned information:

key	type	explanation
fontname	string	the PostScript name of the font
fullname	string	the formal name of the font
familyname	string	the family name this font belongs to
weight	string	a string indicating the color value of the font
version	string	the internal font version
italicangle	float	the slant angle

`luafont, table = fontloader.open(string, [string])` Parse a font file and return a table representing its contents. The optional argument is the name of the desired font in case of font collection files. The optional return value contains any parser error strings.

Listing all of the substructure returned from `fontloader.open` would take too much room, see the big reference manual.

`fontloader.apply_featurefile(luafont, string)` Apply a feature file to a fontloader table.

`fontloader.apply_afmfile(luafont, string)`
 Apply an AFM file to a fontloader table.

Image table

Full list of `(image)` object fields:

field name	type	description
depth	number	the image depth for LuaTeX (in scaled points)
height	number	the image height for LuaTeX (in scaled points)
width	number	the image width for LuaTeX (in scaled points)
transform	number	the image transform, integer number 0..7
attr	string	the image attributes for LuaTeX
filename	string	the image file name
stream	string	the raw stream data for an /XObject /Form object

page	??	the identifier for the requested image page (type is number or string, default is the number 1)
pagebox	string	the requested bounding box, one of none, media, crop, bleed, trim, art
bbox	table	table with 4 boundingbox dimensions lly, lly, urx, and ury overruling the pagebox entry
filepath	string	the full (expanded) file name of the image
colordepth	number	the number of bits used by the color space
colorspace	number	the color space object number
imagetype	string	one of pdf, png, jpg, jbig2, or nil
objnum	number	the pdf image object number
index	number	the pdf image name suffix
pages	number	the total number of available pages
xsize	number	the natural image width
ysize	number	the natural image height
xres	number	the horizontal natural image resolution (in DPI)
yres	number	the vertical natural image resolution (in DPI)

`image = img.new([table])` This function creates an 'image' object. Allowed fields in the table: "filename" (required), "width", "depth", "height", "attr", "page", "pagebox", "colorspace".

`table = img.keys()` Returns a table with possible image table keys, including retrieved information.

`image = img.scan(image)` Processes an image file and stores the retrieved information in the image object.

`image = img.copy(image)` Copy an image.

`image = img.write(image)` Write the image to the PDF file.

`image = img.immediatewrite(image)` Write the image to the PDF file immediately.

`node = img.node(image)` Returns the node associated with an image.

`table = img.types()` Returns a list of supported image types.

`table = img.bboxes()` Returns a list of supported image bounding box names.

Kpathsea table

`kpse.set_program_name(string, [string])`

Initialize the kpathsea library by setting the program name. The optional string allows explicit progname setting.

`kpathsea = kpse.new(string, [string])` Create a new kpathsea library instance. The optional string allows explicit progname setting.

`string = kpse.find_file(string, [string], [boolean], [number])`

Find a file. The optional string is the file type as supported by the standalone kpsewhich program (default is "tex", no autodiscovery takes place). The optional boolean indicates whether the file must exist. The optional number is the dpi value for PK files.

`string = kpse.lookup(string, table)` Find a file (extended interface).

The `kpse.lookup` options match commandline arguments from kpsewhich:

key	type	description
debug	number	set debugging flags for this lookup
format	string	use specific file type (see list above)
dpi	number	use this resolution for this lookup; default 600
path	string	search in the given path
all	boolean	output all matches, not just the first
must-exist	boolean	search the disk as well as ls-R if necessary
mktexpk	boolean	disable/enable mktexpk generation for this lookup
mktxtex	boolean	disable/enable mktxtex generation for this lookup
mktexmf	boolean	disable/enable mktexmf generation for this lookup
mktexfm	boolean	disable/enable mktexfm generation for this lookup
subdir	string or table	only output matches whose directory part ends with the given string(s)

`kpse.init_prog(string, number, string, [string])` Initialize a PK gen-

eration program. The optional string is the metafont mode fallback name

`string = kpse.readable_file(string)` Returns true if a file exists and is readable.

`string = kpse.expand_path(string)` Expand a path.

`string = kpse.expand_var(string)` Expand a variable.

`string = kpse.expand_braces(string)` Expand the braces in a variable.

`string = kpse.show_path(string)` List the search path for a specific file type.

`string = kpse.var_value(string)` Return the value of a variable.

`string = kpse.version()` Return the kpathsea version.

Language table

`language = lang.new([number])` Create a new language object, with an optional fixed id number.

`number = lang.id(language)` Returns the current internal language id number.

`[string] = lang.hyphenation(language, [string])` Get or set hyphenation exceptions.

`lang.clear_hyphenation(language)` Clear the set of hyphenation exceptions.

`string = lang.clean(string)` Creates a hyphenation key from the supplied hyphenation exception.

`[string] = lang.patterns(language, [string])` Get or set hyphenation patterns.

`lang.clear_patterns(language)` Clear the set of hyphenation patterns.

`[number] = lang.prehyphenchar(language, [number])`

Set the pre-hyphenchar for implicit hyphenation.

`[number] = lang.posthyphenchar(language, [number])`

Set the post-hyphenchar for implicit hyphenation.

`[number] = lang.preexhyphenchar(language, [number])`

Set the pre-exhyphenchar for explicit hyphenation.

`[number] = lang.postexhyphenchar(language, [number])`

Set the post-exhyphenchar for explicit hyphenation.

`boolean = lang.hyphenate(node, [node])` Hyphenate a node list.

Lua table

There are 65536 bytecode registers, that are saved in the format file. Assignments are always global.

`function = lua.getbytecode(number)`

Return a previously stored function from a bytecode register.

`lua.setbytecode(number, function)`

Save a function in a bytecode register.

They also be accessed via the virtual array `lua.bytecode[]`.

The virtual array `lua.name[]` can be used to give names to lua chunks. To use `lua.name[1]`, set `lua.name[1] = 'testname'` and `directlua1[rubbish]`.

Metapost table

`string = mplib.version()` Returns the mpplib version.

`mpinstance = mpplib.new(table)` Create a new metapost instance.

`mpdata = mp.execute(string)` Execute metapost code in the instance.

`mpdata = mp.finish()` Finish a metapost instance.

The return value of `mp.execute` and `mp.finish` is a table with a few possible keys (only status is always guaranteed to be present).

log	string	output to the 'log' stream
term	string	output to the 'term' stream
error	string	output to the 'error' stream (only used for 'out of memory')

status	number	the return value: 0=good, 1=warning, 2=errors, 3=fatal error
--------	--------	--

fig	table	an array of generated figures (if any)
-----	-------	--

Handling of fig objects would take too much room here, please see the big reference manual.

`table = mp.statistics()` Returns some statistics for this metapost instance.

`number = mp.char_width(string, number)` Report a character's width.

number = **mp.char_height**(string, number) Report a character's height.
 number = **mp.char_depth**(string, number) Report a character's depth.

Node table

table = **node.types**() Return the list of node types.
 table = **node.whatsits**() Return the list of whatsit types.
 number = **node.id**(string) Convert a node type string into a node id number.
 number = **node.subtype**(string) Convert a whatsit type string into a node subtype number.
 string = **node.type**(number) convert a node id number into a node type string.
 table = **node.fields**(number, [number]) Report the fields a node type understands. The optional argument is needed for whatsis.
 boolean = **node.has_field**(node, string) Return true if the node understands the named field.
 node = **node.new**(number, [number]) Create a new node with id and (optional) subtype.
node.free(node) Release a node.
node.flush_list(node) Release a list of nodes.
 node = **node.copy**(node) Copy a node.
 node = **node.copy_list**(node, [node]) Copy a node list.
 node, number = **node.hpack**(node, [number], [string], [string]) Pack a node list into a horizontal list. The number is the desired size, the first string is either "exact" or "additional" (modifies the first string), the second string is the desired direction
 node, number = **node.vpack**(node, [number], [string], [string]) Pack a node list into a vertical list. Arguments as for node.hpack
 number, number, number = **node.dimensions**([number], [number], [number], node, [node])
 Return the natural dimensions of a (horizontal) node list. The 3 optional numbers represent glue_set, glue_sign, and glue_order. The calculation stops just before the optional node (default end of list)
 node = **node.mlist_to_hlist**(node, string, boolean) Recursively convert a math list into a horizontal list. The string differentiates display and inline, the boolean whether penalties are inserted
 node = **node.slide**(node) Move to the last node of a list while fixing next and prev pointers.
 node = **node.tail**(node) Return the last node in a list.
 number = **node.length**(node, [node]) Return the length of a node list. Processing stops just before the optional node.
 number = **node.count**(number, node, [node])
 Return the count of nodes with a specific id in a node list. Processing stops just before the optional node.
 node = **node.traverse**(node) Iterate over a node list.
 node = **node.traverse_id**(number, node)
 Iterate over nodes with id matching the number in a node list.
 node, node = **node.remove**(node, node) Extract and remove a second node from the list that starts in the first node.
 node, node = **node.insert_before**(node, node, node) Insert the third node just before the second node in the list that starts at the first node.
 node, node = **node.insert_after**(node, node, node)
 Insert the third node just after the second node in the list that starts at the first node.
 node = **node.first_character**(node, [node]) Return the first character node in a list. Processing stops just before the optional node.
 node, node, boolean = **node.ligaturing**(node, [node])
 Apply the internal ligaturing routine to a node list. Processing stops just before the optional node.
 node, node, boolean = **node.kerning**(node, [node])
 Apply the internal kerning routine to a node list. Processing stops just before the optional node.
 node, node, boolean = **node.unprotect_glyphs**(node) Mark all characters in a node list as being processed glyphs.
 node, node, boolean = **node.protect_glyphs**(node) Mark all processed glyphs in a node list as being characters.

node = **node.last_node**() Pops and returns the last node on the current output list.
 node, **node.write**(node) Appends a node to the current output list.
 boolean = **node.protrusion_skippable**(node) Return true if the node could be skipped for protrusion purposes.
 number = **node.has_attribute**(node, number, [number]) Return an attribute value for a node, if it has one. The optional number tests for a specific value
node.set_attribute(node, number, number) Set an attribute value for a node.
 number = **node.unset_attribute**(node, number, [number])
 Unset an attribute value for a node. The optional number tests for a specific value

Pdf table

number = **pdf.immediateobj**([number], [string], string, [string])
 Write an object to the PDF file immediately. The optional number is an object id, the first optional string is "file", "stream", or "filestream", the second optional string contains stream attributes for the latter two cases.
pdf.mapfile(string) Register a font map file.
pdf.mapline(string) Register a font map line.
 number = **pdf.obj**([number], [string], string, [string]) Write an object to the PDF file. See "pdf.immediateobj" for arguments.
 number = **pdf.pagerref**(number) Return the pagerref object number.
pdf.print([string], string)
 Write directly to the PDF file (use in \LaTeX). The optional string is one of "direct" or "page"
 number = **pdf.reserveobj**()
 Reserve an object number in the PDF backend.
pdf.registerannot(number) Register an annotation in the PDF backend.

Status table

table = **status.list**() Returns a table with various status items.
 The current list is:

key	explanation
pdf_gone	written pdf bytes
pdf_ptr	not yet written pdf bytes
dvi_gone	written dvi bytes
dvi_ptr	not yet written dvi bytes
total_pages	number of written pages
output_file_name	name of the pdf or dvi file
log_name	name of the log file
banner	terminal display banner
var_used	variable (one -word) memory in use
dyn_used	token (multi -word) memory in use
str_ptr	number of strings
init_str_ptr	number of \TeX strings
max_strings	maximum allowed strings
pool_ptr	string pool index
init_pool_ptr	\TeX string pool index
pool_size	current size allocated for string characters
node_mem_usage	a string giving insight into currently used nodes
var_mem_max	number of allocated words for nodes
fix_mem_max	number of allocated words for tokens
fix_mem_end	maximum number of used tokens
cs_count	number of control sequences
hash_size	size of hash
hash_extra	extra allowed hash
font_ptr	number of active fonts
max_in_stack	max used input stack entries
max_nest_stack	max used nesting stack entries
max_param_stack	max used parameter stack entries
max_buf_stack	max used buffer stack entries
max_save_stack	max used save stack entries
stack_size	input stack size

nest_size nesting stack size
param_size parameter stack size
buf_size current allocated size of the line buffer
save_size save stack size
obj_ptr max pdf object pointer
obj_tab_size pdf object table size
pdf_os_cnr max pdf object stream pointer
pdf_os_objidx pdf object stream index
pdf_dest_names_ptr max pdf destination pointer
dest_names_size pdf destination table size
pdf_mem_ptr max pdf memory used
pdf_mem_size pdf memory size
largest_used_mark max referenced marks class
filename name of the current input file
inputid numeric id of the current input
linenumber location in the current input file
lasterrorstring last error string
lua_bytecodes number of active Lua bytecode registers
lua_bytecodes_bytes number of bytes in Lua bytecode registers
luastate_bytes number of bytes in use by Lua interpreters
output_active true if the \output routine is active
callbacks total number of executed callbacks so far
indirect_callbacks number of those that were themselves a result of other callbacks (e.g. file readers)

luatex_svn the luatex repository id (added in 0.51)
luatex_version the luatex version number (added in 0.38)
luatex_revision the luatex revision string (added in 0.38)
ini_version true if this is an iniT_EX run (added in 0.38)

Typesetting table

tex.set([string], string, value) Set a named internal register. Also accepts a predefined csname string.

value = **tex.get**(string) Get a named internal register. Also accepts a predefined csname string.

Many of LuaT_EX's internal parameters can be queried and set this way, but not nearly all. The big reference manual has an extensive list.

tex.setattribute([string], number, number)

Set an attribute register. Also accepts a predefined csname string.
number = **tex.getattribute**(number)

Get an attribute register. Also accepts a predefined csname string.

tex.setbox([string], number, node) Set a box register. Also accepts a predefined csname string.

node = **tex.getbox**(string) Get a box register. Also accepts a predefined csname string.

tex.setcount([string], number, number)

Set a count register. Also accepts a predefined csname string.

number = **tex.getcount**(number) Get a count register. Also accepts a predefined csname string.

tex.setdimen([string], number, number)

Set a dimen register. Also accepts a predefined csname string.

number = **tex.getdimen**(number) Get a dimen register. Also accepts a predefined csname string.

tex.setskip([string], number, node) Set a skip register. Also accepts a predefined csname string.

node = **tex.getskip**(number)

Get a skip register. Also accepts a predefined csname string.

tex.settoks([string], number, string) Set a toks register. Also accepts a predefined csname string.

string = **tex.gettoks**(number)

Get a toks register. Also accepts a predefined csname string.

tex.setcatcode([string], [number], number, number)

Set a category code.

number = **tex.getcatcode**([number], number) Get a category code.

tex.setlccode([string], number, number, [number])

Set a lowercase code.

number = **tex.getlccode**(number) Get a lowercase code.

tex.setsfcodes([string], number, number) Set a space factor.

number = **tex.getsfcodes**(number) Get a space factor.

tex.setuccode([string], number, number, [number]) Set an uppercase code.

number = **tex.getuccode**(number) Get an uppercase code.

tex.setmathcode([string], number, table) Set a math code.

table = **tex.getmathcode**(number) Get a math code.

tex.setdelcode([string], number, table) Set a delimiter code.

table = **tex.getdelcode**(number) Get a delimiter code.

In all the **tex.set...** functions above, the optional string is the literal "global".

The items can also be accessed directly via virtual arrays: **tex.attributes**[], **tex.box**[], **tex.count**[], **tex.dimen**[], **tex.skip**[], **tex.toks**[], **tex.catcode**[], **tex.lccode**[], **tex.sfcodes**[], **tex.uccode**[], **tex.mathcode**[], **tex.delcode**[],

tex.setmath([string], string, string, number)

Set an internal math parameter. The first string is like the csname but without the Umath prefix, the second string is a style name minus the style suffix.

number = **tex.getmath**(string, string) Get an internal math parameter.

The first string is like the csname but without the Umath prefix, the second string is a style name minus the style suffix.

tex.print([number], string, [string]) Print a sequence of strings (not just two) as lines. The optional argument is a catcode table id.

tex.sprint([number], string, [string]) Print a sequence of strings (not just two) as partial lines. The optional argument is a catcode table id.

tex.tprint(table, [table]) Combine any number of **tex.sprint**'s into a single function call.

tex.write(string) Print a sequence of strings (not just two) as detokenized data.

number = **tex.round**(number) Round a number.

number = **tex.scale**(number, number) Multiplies the first number (or all fields in a table) with the second argument (if the first argument is a table, so is the return value).

number = **tex.sp**(string) Convert a dimension string to scaled points.

tex.definfont([boolean], string, number)

Define a font csname. The optional boolean indicates for global definition, the string is the csname, the number is a font id.

tex.error(string, [table]) Create an error that is presented to the user.

The optional table is an array of help message strings.

tex.enableprimitives(string, table)

Enable the all primitives in the array using the string as prefix.

table = **tex.extraprimitives**(string, [string]) Return all primitives in a (set of) extension identifiers. Valid identifiers are: "tex", "core", "etex", "pdftex", "omega", "aleph", and "luatex".

table = **tex.primitives**() Returns a table of all currently active primitives, with their meaning.

number = **tex.badness**(number, number) Compute a badness value.

tex.linebreak(node, table) Run the line breaker on a node list. The table lists settings.

The **tex.linebreak** parameters:

name	type	description
pardir	string	
pretolerance	number	
tracingparagraphs	number	
tolerance	number	
looseness	number	
hyphenpenalty	number	
exhyphenpenalty	number	
pdfadjustspacing	number	
adjdemerits	number	
pdfprotrudechars	number	
linepenalty	number	
lastlinefit	number	
doublehyphendemerits	number	
finalhyphendemerits	number	
hangafter	number	

interlinepenalty	number or table	if a table, then it is an array like <code>\interlinepenalties</code>
clubpenalty	number or table	if a table, then it is an array like <code>\clubpenalties</code>
widowpenalty	number or table	if a table, then it is an array like <code>\widowpenalties</code>
brokenpenalty	number	
emergencystretch	number	in scaled points
hangindent	number	in scaled points
hsize	number	in scaled points
leftskip	glue_spec node	
rightskip	glue_spec node	
pdfeachlineheight	number	in scaled points
pdfeachlinedepth	number	in scaled points
pdffirstlineheight	number	in scaled points
pdfastlinedepth	number	in scaled points
pdfignoreddimen	number	in scaled points
parshape	table	

The `tex.linebreak` returned table data:

<code>prevdepth</code>	depth of the last line in the broken paragraph
<code>prevgraf</code>	number of lines in the broken paragraph
<code>looseness</code>	the actual looseness value in the broken paragraph
<code>demerits</code>	the total demerits of the chosen solution

`tex.shipout(number)` Ships the box to the output file and clears the box.

The virtual table `tex.lists` contains the set of internal registers that keep track of building page lists.

field	description
<code>page_ins_head</code>	circular list of pending insertions
<code>contrib_head</code>	the recent contributions
<code>page_head</code>	the page-so-far
<code>hold_head</code>	used for held-over items for next page
<code>adjust_head</code>	head of the current <code>\adjust</code> list
<code>pre_adjust_head</code>	head of the current <code>\adjust pre</code> list

The virtual table `tex.nest` contains the currently active semantic nesting state. It has two main parts: an zero-based array of userdata for the semantic nest itself, and the numerical value `tex.nest.ptr`. Known fields:

key	type	modes	explanation
<code>mode</code>	number	all	The current mode. 0 = no mode, 1 = vertical, 127 = horizontal, 253 = display math. -1 = internal vertical, -127 = restricted horizontal, -253 = inline math.
<code>modeline</code>	number	all	source input line where this mode was entered in, negative inside the output routine.
<code>head</code>	node	all	the head of the current list
<code>tail</code>	node	all	the tail of the current list
<code>prevgraf</code>	number	<code>vmode</code>	number of lines in the previous paragraph
<code>prevdepth</code>	number	<code>vmode</code>	depth of the previous paragraph
<code>spacefactor</code>	number	<code>hmode</code>	the current space factor
<code>dirs</code>	node	<code>hmode</code>	internal use only
<code>noad</code>	node	<code>mmode</code>	internal use only
<code>delimpr</code>	node	<code>mmode</code>	internal use only
<code>mathdir</code>	boolean	<code>mmode</code>	true when during math processing the <code>\mathdir</code> is not the same as the surrounding <code>\textdir</code>
<code>mathstyle</code>	number	<code>mmode</code>	the current <code>\mathstyle</code>

Texconfig table

This is a table that is created empty. A startup Lua script could fill this table with a number of settings that are read out by the executable after loading and executing the startup file.

key	type	default	explanation
<code>kpse_init</code>	boolean	trac	false totally disables <code>kpath-sea</code> initialisation
<code>shell_escape</code>	string		cf. <code>web2c docs</code>
<code>shell_escape_commands</code>	string		cf. <code>web2c docs</code>
<code>string_vacancies</code>	number	75000	cf. <code>web2c docs</code>
<code>pool_free</code>	number	5000	cf. <code>web2c docs</code>
<code>max_strings</code>	number	15000	cf. <code>web2c docs</code>
<code>strings_free</code>	number	100	cf. <code>web2c docs</code>
<code>nest_size</code>	number	50	cf. <code>web2c docs</code>
<code>max_in_open</code>	number	15	cf. <code>web2c docs</code>
<code>param_size</code>	number	60	cf. <code>web2c docs</code>
<code>save_size</code>	number	4000	cf. <code>web2c docs</code>
<code>stack_size</code>	number	300	cf. <code>web2c docs</code>
<code>dvi_buf_size</code>	number	16384	cf. <code>web2c docs</code>
<code>error_line</code>	number	79	cf. <code>web2c docs</code>
<code>half_error_line</code>	number	50	cf. <code>web2c docs</code>
<code>max_print_line</code>	number	79	cf. <code>web2c docs</code>
<code>hash_extra</code>	number	0	cf. <code>web2c docs</code>
<code>pk_dpi</code>	number	72	cf. <code>web2c docs</code>
<code>trace_file_names</code>	boolean	trac	false disables \TeX 's normal file feedback
<code>file_line_error</code>	boolean	false	<code>file:line style error</code> messages
<code>halt_on_error</code>	boolean	false	abort run on the first encountered error
<code>formatname</code>	string		if no format name was given on the command-line, this will be used as <code>formatname</code> .
<code>jobname</code>	string		

IO table

`texio.write([string], string)` Write a string to the log and/or terminal. The optional argument is "term", "term and log", or "log".

`texio.write_nl([string], string)` Write a string to the log and/or terminal, starting on a new line. The optional argument is "term", "term and log", or "log".

Token table

A token is represented in Lua as a small table. For the moment, this table consists of three numeric entries:

index	meaning	description
1	command code	this is a value between 0 and 130
2	command modifier	this is a value between 0 and 2 ²¹
3	control sequence id	for commands that are not the result of control sequences, like letters and characters, it is zero, otherwise, it is a number pointing into the 'equivalence table'

`token = token.get_next()` Fetch the next token from the input stream.

`boolean = token.is_expandable(token)`

True if the token is expandable.

`token.expand()`

Expand a token the tokenb waiting in the input stream.

`boolean = token.is_activechar(token)`

True if the token represents and active character.

`token = token.create(number, [number])` Create a token from scratch, the optional argument is a category code. Also accepts strings, in which case a token matching that `csname` is created.

`string = token.command_name(token)`

Return the internal string representing a command code.

`number = token.command_id(string)`

Return the internal number representing a command code.

`string = token.csname_name(token)` Return the `csname` associated with a token.

`number = token.csname_id(string)` Returns the value for a `csname` string.

Subtext: A Proposed Processual Grammar for a Multi-Output Pre-Format

Subtext: návrh postupové gramatiky na předformát určený pro různá výstupní zařízení

JOHN HALTIWANGER

Abstract: Academic publishing today faces a reality in which providing multiple formats—generally HTML and PDF—is becoming a necessity. The production of multiple outputs involves a workflow of *generative typesetting*. Generative typesetting involves many constraints that resulting from edge cases between formats which must be accounted for. Against the backdrop of theory in the field of new media, a new approach towards generative typesetting is proposed. A separation of translation from effect, akin to the division of style and content in HTML/CSS, can effect a *transmutable translation* layer in which syntax, effect, and even a pre-format’s reserved characters can be defined in configuration files. This transmutability is desirable because every generative typesetting workflow faces particular specificities which should be addressable without the introduction of “glue.”

Keywords: typesetting, processual grammar, pre-format, multi-output.

Abstrakt: Článek přináší několik myšlenek a úvah na návrh gramatiky pro předformát u vícenásobného výstupu z jednoho zdrojového kódu.

Klíčová slova: sazba, typografie, postupová gramatika, předformát, vícenásobný výstup.

Introduction

Over the course of my recent work obtaining a master’s degree in New Media at the Universiteit van Amsterdam, I had the opportunity to investigate the field of generative design—and especially generative typesetting—in the context of a thesis that interrogates existing media theories for their efficacy in describing the dynamics involved in generative workflows.

The core impulse of my thesis began with the idea of a web application which offers a simple means of ‘writing once, reading everywhere.’ In terms of

the practical problem domain (academic publishing of humanities texts), this meant targetting two formats: HTML and PDF. There seemed to be interesting questions of ‘materiality’ in any discussion of the envisioned workflow:

1. A pre-format in which the source document is written. (For my thesis, I chose Markdown.) What *material specificities* does such a format exhibit? From a less directed orientation, what does it mean to write in a format whose sole purpose is to be translated into *other* formats?
2. A ‘wrapper’ (or translation layer) application. (For now, in terms of ConTeXt, that means Pandoc.) What does it say about the ‘materiality’ of the final product when, indeed, the final “product” is a multitude of output files (PDF for print, PDF for screen, HTML for browser, HTML for handheld, etc.)?
3. The glue layer. This layer was recognized as inevitable from the outset due to both personal conversations with people who have attempted cross-format publishing and the knowledge that with (translation) software, it will never Just Work Right. The ‘materiality’ of the outputs further spreads out across code written to satisfy edge cases and account for inadequacies in either of the other two layers.

The Interest in Materiality

Recently there has been a shift within the field of new media back towards an adherence to Marshall McLuhan’s old motto: “the medium is the message.” This shift has been very productive in its inspiration towards *software studies*, *medium-specific analysis*, and a host of other approaches of examining media as media. While quite productive in generating theory, this shift has not necessarily been so productive in terms of deciphering the (often unasked) question: What defines a medium?

The contemporary fixation with ‘materiality’ is a result of seeking the constitution of media from inside themselves, a fact that belies a neglect in attention to the shaping of media by processes external to themselves such as economics, political climate (including legal structures), history, and imagination.

It’s All Process: A Shift to an Analytics of Becoming

There may be a better way to critically engage with media, one that can account for and respond to the fragmentation that the concept of ‘medium’ has experienced since the advent of the personal computer. Though relatively obscure in English language discourse, Gilbert Simondon’s theory of *ontogenesis* is remarkably applicable for describing the unfolding of structures within the computer metamedium.¹

¹Loosely translatable as a machine in which the available potential of the machine is definable from within that machine itself, a feature that lets the computer “play host” to other media.

Simondon's ontogenesis transcends the traditional Western conundrum of existence ("what am I? what does it mean that I am?") by asking the question: "How did I become? What am I becoming?"

This is a radical shift, as the disconnect between the conditions that shape us and our current form is missing from many fields—the most obvious example lying in the forms of economic analysis that refuse to integrate so-called "externalities" such as vertical market dynamics, resource depletion, pollution, and the general turmoil associated with extravagantly rich men getting richer while poor people get poorer. In the field of new media, specifically, it allows for a *process-oriented* perspective through which the conditions of a given medium can be integrated into our understanding of that medium. In a way it is not a rejection of 'looking into the medium' for understanding; rather, it implores an additional *looking around*.

Source Code as a Site of Collective Agency

From the perspective of metamedia, where constraints are defined by programming as much as by the physical properties of the machine, source code becomes a site of agency: if one can rule the source, one can rule the metamedium. Through an examination of the discussion between Hans Magnus Enzensberger and Jean Baudrillard regarding a revolutionary theory of media, I position FLoSS as a unique and potent site of agency within metamedia (especially the still-dominant personal computer). FLoSS not only fits the seven categories of emancipatory uses of media outlined by Enzensberger, it likewise operates in a "reciprocal" (actively reflexive) way that restores the symbolic exchange relation that Baudrillard deems necessary for any subversive potential of media.

Tied to the concept of ontogenesis, FLoSS becomes a vast site of collective becoming within the computer metamedium. While many people are squeamish about granting it this mantle, the truth remains that FLoSS, and only FLoSS, carries within itself the potential for total and radical change of the metamedium. Likewise, only FLoSS provides a means for hedging against the increasing lock-in of metamedia that we find in devices such as the iPhone and the Kindle. These features are significant and the stakes are real: metamedia could cease to be metamedia if it becomes a common-place assumption that one is not intended to program them on one's own and without following strict rules of what can be programmed and how. This is a dangerous path, yet the rapid adoption of the iPhone and iPad—despite the valid criticism that the devices rigidly enforce a crippled, consumerist take on computing—proves that this direction worth examining as a very real potential road for the future of social understandings of computing.

Processual Grammars Organize Process Hybridity

A very real problem in terms of ‘medium-specific analysis’ that we find when dealing with computers is that there can be distinct specificities even between different versions of the *same program*. Does each new or distinct version constitute a specific medium? Just as the variety of typewriter makes and models do not constitute individual media, it makes no sense to apply the ‘medium’ label to every application on the computer. Yet there are distinct new capabilities and processes embodied within computer programs, and it seems just as nonsensical to render everything available on the computer as belonging to, or representing, a single medium.

Through an analytics of becoming we are made aware of the various hybridizations involved in the assembling of processes. Take, for example, the modern command-line interface (CLI). The original roots of the CLI lie in the teletype machines, which were quite literally the hybridization of typewriters and computer feedback systems such as tape or punch cards, along with telephone and modem technology. Today there are often other aspects in play: a framebuffer (perhaps even a GUI, in the case of terminal emulators), along with libraries that handle text not represented directly on the keyboard, as well as continuously evolving shells and terminal emulators.

What I argue is that these various configurations—resulting from evolutionary developments in and outside the CLI—represent distinct *processual grammars* that organize various processes into distinct hybridities. Thus the differences between the bash and csh shells (or between GIMP and Photoshop) indicate not the existence of a variety of separate media but rather divergent organizational grammars of process that engender intentional specificities that are designed to overcome or otherwise modify some real, existing constraint. In the case of bash versus csh, the constraint was ease of programming and in the case of the GIMP it was the lack of a free software alternative for hardcore raster image manipulation. The significance of FLoSS in this light is that, thanks to the availability of source code, we have the capacity to modify, extend, or replace processual grammars at will. This is something ultimately familiar to the T_EX community, which has seen the steady progression of engines and patches intended to accommodate and account for overcoming constraints within the original implementation. If T_EX’s entire internals were proprietary and the source code withheld, these efforts would require a completely different level of work.²

²This is also assuming that Donald Knuth had not been so forthright in his writings on the topic due to proprietary concern for ‘trade secrets.’

The Problem Domain

Academic papers in the humanities are still written in WYSIWYG word processors, most often the proprietary Microsoft Word. This is pervasive to the extent that editing jobs require the possession of this software. Yet the chief feature of that software in this problem domain is the “Track Changes” mode—something which version control systems handle seamlessly with regards to plain-text documents. The issue is not technical, then, but a question of interface. T_EX is a great system, but it is still a visually cumbersome programming language. An abstraction into a pre-format for interfacing with typesetting could allow the adoption of T_EX-based workflows without the necessity of learning T_EX (or L^AT_EX or ConT_EXt). A new interface for version control based around the workflow demands of publishing and editing would likewise smooth a transition from proprietary conditions.

Requiring proprietary software for academic publishing is a strange product of history and economics. Both of these influences have been known to be overcome through FLoSS. Indeed, history and economics were both overcome when Richard Stallman made an institution and a legal hack around the considered-obsolete institution of code sharing. It is possible to imagine a new system for generative typesetting that outshines previous workflows of academic writing.³

Towards A New Processual Grammar for Typesetting

My thesis attempts an actively reflexive methodology: in order to investigate generative design, a workflow based on generative typesetting was employed. This workflow has already been outlined in the introduction to this piece: it is composed of the *pre-format*, the *translation layer*, and the *glue layer*.

It seems to me that with the advent of LuaT_EX we now have the opportunity to integrate these three layers and, while doing so, modify them in order to fulfill our expectations in a more efficacious manner. Since LuaT_EX allows for the utilization of Lua code, we can theoretically implement anything possible in Lua as an add-on for LuaT_EX.

The first stage of this is to re-think the pre-format. I define a pre-format as a markup that is chiefly designed both for ease of *writing* and for *translatability*. In other words, pre-formats exist to provide a drastic reductions in learning-curve and syntactical verbosity. However, prominent pre-formats like MARKDOWN and RESTRUCTUREDTEXT were designed primarily by programmers for programmers. In other words, the factors involved in typesetting are not always accounted for in these pre-formats. It is time to change this. We

³Even if the system outlined here is not ideal, I maintain that there has to be a better way of dealing with publishing in the humanities.

need a pre-format that understands how to pass environment information so that a paragraph, for example, can be marked as a member of an arbitrary environment (or class/id in HTML). Ideally all markup should be augmentable with contextual specificities that then map into the outputs. A huge element of this hybridization is delivering bibliographic/citation functionality that extends to all outputs. Such a hybridization—not currently possible in Pandoc—would elevate this processual grammar above the competition (for instance, Bib $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ requires significant massaging just to output the MLA format, a standard in the humanities). Since the translation layer can be implemented in Lua, it is possible to develop a library for Lua $\text{T}_{\text{E}}\text{X}$ that processes this pre-format and output the results in multiple other formats.⁴ By hybridizing the translation layer into Lua $\text{T}_{\text{E}}\text{X}$, beautiful typesetting into PDFs will become a first-order capability of the pre-format. Perhaps the best way to phrase this concept to developers is that it represents a recognition that a *pre-format can be built as a domain-specific language*. To that extent, the syntax, its effects and even the reserved characters can be specified through configuration files and metaprogramming. This allows for personally tailored workflows as well as defined and emergent standards of operating.

Glue-Be-Gone

This could theoretically represent a point of transcendence of that final layer of integration: the glue layer. An *transmutable* translation layer allows a new separation of translation from implementation, something akin to the attempted division of form and content vis a vis CSS and HTML. As mentioned, in this scheme the exact effects of the pre-format syntax (i.e., what the translator outputs upon receiving a certain markup) can be defined on a per-project (or even per-individual) basis. Pandoc, on the other hand, allows for scripting but only in a way that involves writing Haskell code. While this may be fine in some cases, clearly the necessity of learning a new programming language simply to use a translation wrapper is less-than-ideal. By loading rules in from configuration files we can separate the workflow into subsystems that align to different degrees of engagement with the workflow: users can easily interact within the specific syntax of a project, yet they can just as easily begin to modify that syntax by changing values in a configuration file. If they are interested in engaging the outputs more deeply, they can alter other configuration values in order to substitute their own values as effects of translation. The point is to allow multiple levels of affecting how *this* gets changed into *that*.

The system need not necessarily be designed for inter-operability between

⁴Another option, worth investigating, is using the powerful grammar processing of Perl 6 in combination with the Parrot Virtual Machine and its native Lua implementation.

specific organizations of workflow. However, even that goal should be relatively accessible if shared code works only on an abstract level and respects arguments such as environments that have been passed from the pre-format. These arguments will have a defined structure within the configuration files, so errors in syntax can be reported when they are found during translation and even shared code libraries can be written to take these structures into account. I am not saying that edge cases will never occur, only that they should be adjustable on a granular level that allows a single workflow to be optimized and *just work*. (These adjustments will require engaging with environment setup through mechanisms such as CSS or ConT_EXt macros). One could imagine a university department developing a tailored pre-format that suits their field and generates arbitrarily optimal PDFs and HTML files, something akin to the current prevalence of L^AT_EX document classes in mathematics and science but with the ease of use and translatability offered by a pre-format.

This is not to say that “glue” will never be necessary, only that we should recognize its perpetual existence in generative typesetting for multiple outputs. As such, it makes sense to mitigate the source of glue, which can be found in the edge cases that must be accounted for between output formats in any given workflow. The uniqueness of every workflow means that the best solution lies in a system sufficiently flexible enough to accommodate arbitrary demands and inevitable weirdness. While adherence to a standard syntax is possible, there should ultimately be as few constraints placed upon the execution of that workflow as possible. As long as the configuration files are available, the system should be able to render any output formats according to the logic declared therein. There is a distinct opportunity to provide a new alternative to academic writing in the humanities. LuaT_EX offers a significant site of agency for delivering on this opportunity. While it is obvious that this proposal consists only of theoretical speculation, I hope that the ideas outlined in this text provide a thought-provoking outlook on conceiving of this delivery.

*john (dot) haltiwanger (at) gmail (dot) com
Amsterdam, The Netherlands*

Abstract: There is still a lot to be considered until we can hold a finished book in our hands, after the content is ready. In this article an overview on possible page arrangement schemes is presented. Although ConT_EXt already has a considerable range of possibilities built in, more arranging schemes will be added in the near future, in order to make ConT_EXt even more versatile.

Key words: Page arrangement, section, booklet, flyer.

Abstrakt: V článku jsou představeny základní způsoby vyřazení dokumentu, což je nezbytná operace před tiskem podkladu. Článek se zaměřuje na formát ConT_EXt a slibuje i další nastavení v době budoucí.

Klíčová slova: Vyřazování dokumentu, vyřazení stránek, příprava knižního bloku, brožurka, pozvánka, leták, pozvánky.

w (dot) egger (at) boede (dot) nl
Maasstraat 2, 5836 BB Sambeek, The Netherlands

1 Arranging pages

Once a document is made up properly, it needs to be put on paper in order to be able to prepare a book. A book can be printed on loose sheets and the book block can then be glued at the spine. The result will often be presented as a paperback. If the book is to be sewn i.e. the book block is to be made up from sections which are sewn together, then the pages of the book must be arranged in such a way, that they can be folded to form the sections.

ConT_EXt is equipped with a set of page imposition schemes which are used in professional printing environments.

In order to describe the different schemes it is a necessary to divide them into different categories.

- Page arrangements for section printing
- Page arrangements for glued bookblocks
- Specials
- Flyers

It must be mentioned, that some of the described schemes are not yet belonging to the ConT_EXt distribution. They will be added in due time.

2 Page arrangements for section printing

Coding of the page arrangements

In order to be able to use page arrangement schemes one needs to tell ConT_EXt what it should do. The information is normally given with two to four pieces of information, separated with *.

The first value, which is 1 or 2, indicates whether the arrangement will be single-sided (1) or double-sided (2).

The second value is always an even number ranging from 2 to 16. This figure indicates the number of pages on the recto respectively on the verso side of the sheet of paper.

The third value indicates the number of sheets to be used to make up a single section.

There are arranging schemes which use another indicator in order to make them unique in the system. Usually this is one or more characters.

Classical sections for large paper sizes

Context standard

2*16 (32 pages) i.e. 2 = doublesided, 16 = 16 pages on recto and 16 pages on verso of the sheet

2*8 (16 pages) i.e. 2 = doublesided, 8 = 8 pages on recto and 8 pages on verso of the sheet

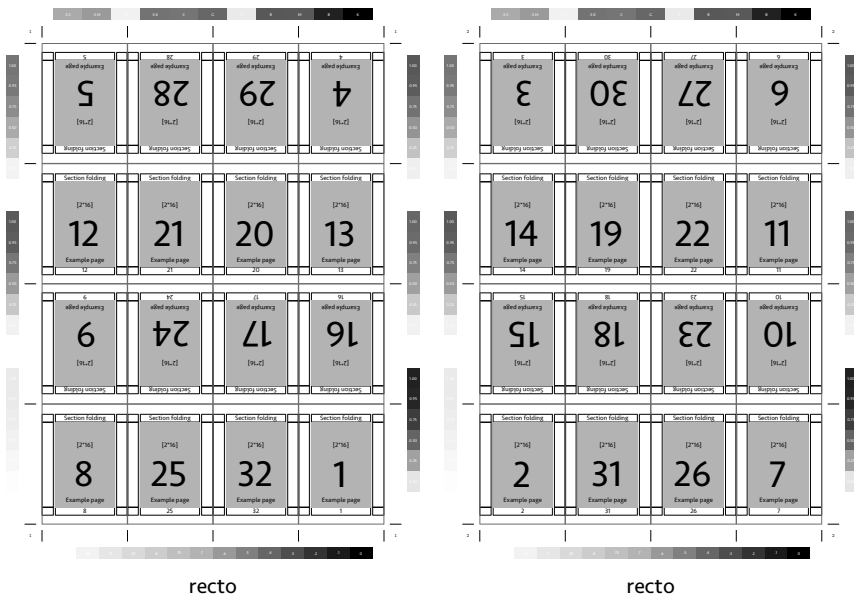


Figure 1 2*16, (32 pages)

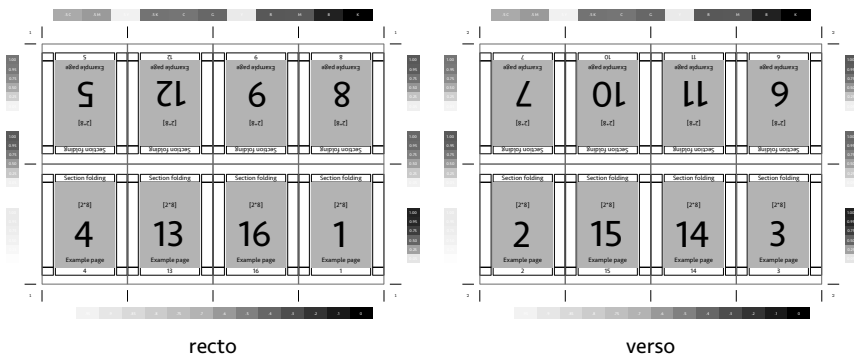


Figure 2 2*8, (16 pages)

Z-folded sections

Besides the classical folding schemes which are based on cross folding a sheet of paper, it is also possible to make sections with Z-folds and a final fold in the direction of the spine. The following schemes are currently not yet in the distribution.

2*6*Z 2 = doubled-sided, 6 = 6 pages on recto and 6 pages on verso, Z = Z-folding

2*8*Z 2 = doubled-sided, 8 = 8 pages on recto and 8 pages on verso, Z = Z-folding

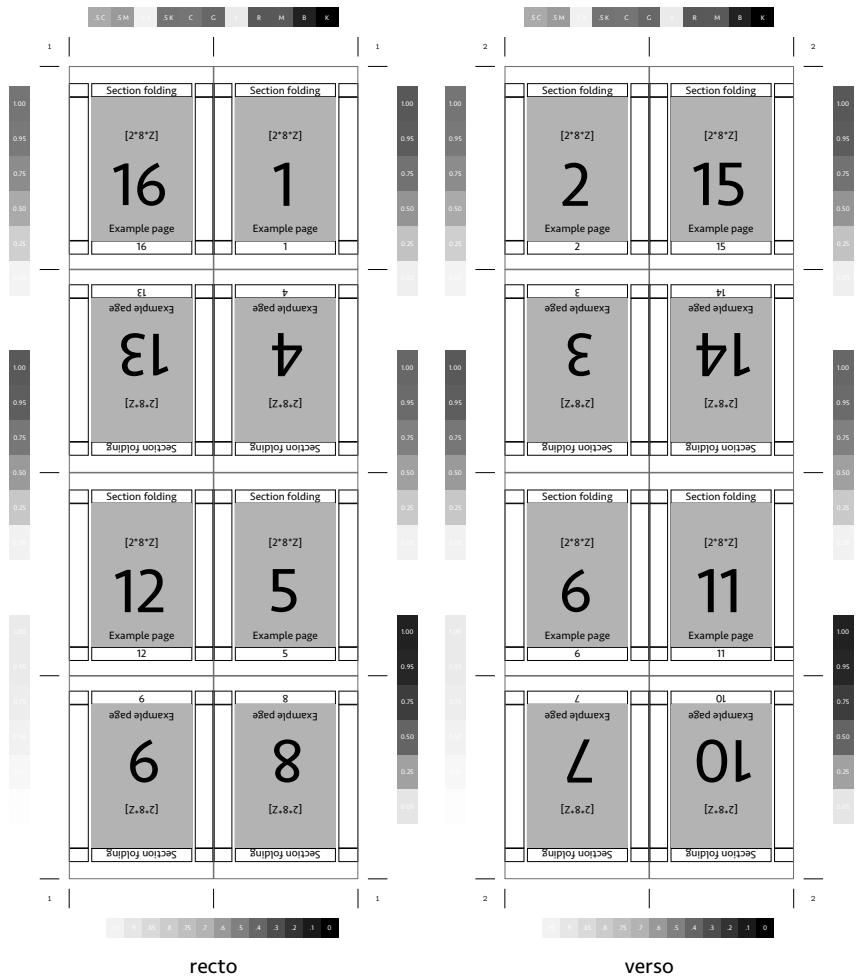


Figure 3 2*8*Z, (16 pages)

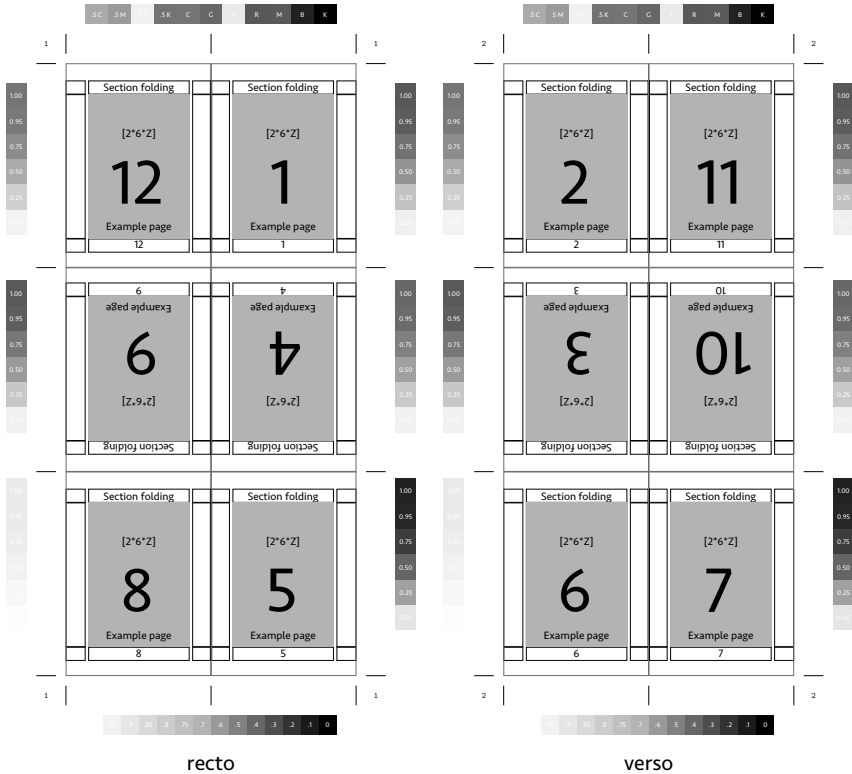


Figure 4 2*6*Z, (12 pages)

Folded sections composed from multiple sheets

Although it is quite uncommon to keep a large schale printer at the office, it is still quite possible to use a normal office-printer to produce classical folded sections for very small numbers of books. In order to get folded sections more than one sheet is used to assemble a single section. The following schemes will be added to the distribution.

2*2*4 2 = doublesided, 2 = 2 pages recto and verso, 4 = 4 sheets of paper

2*4*2 2 = doublesided, 4 = 4 pages recto and verso, 2 = 2 sheets of paper

The classical booklet

Although the classical booklet is not a classical section it is mentioned here. To form a classical booklet ConTeXt assembles all present pages in a document into one (large) section.

- 2DOWN binding at the short edge: 2 = doublesided, DOWN= placement of the pages on the sheet of paper
- 2UP binding at the long edge: 2 = doublesided, UP= placement of the pages on the sheet of paper

3 Page arrangements for glued book blocks

As mentioned above, one can produce a book block from loose sheets of paper without folding them. The sheets can be printed either single or double-sided.

Single-sided schemes

- 1*4 1 = single-sided, 4 pages on recto
- 2SIDE single-sided 2 pages next to each other on recto
- 2TOP single-sided 2 pages on top and bottom on recto
- XY pages are arranged in columns (x) and rows (y)

Double-sided scheme

- 2TOPSIDE double-sided 2 pages recto and verso, top and bottom

4 Specials

The following scheme will be added to the distribution. With this scheme one can prepare a single-sided print and fold it into a booklet in such a way, that the blank verso side is hidden.

- 1*8 1= single-sided, 8 = 8 pages on recto

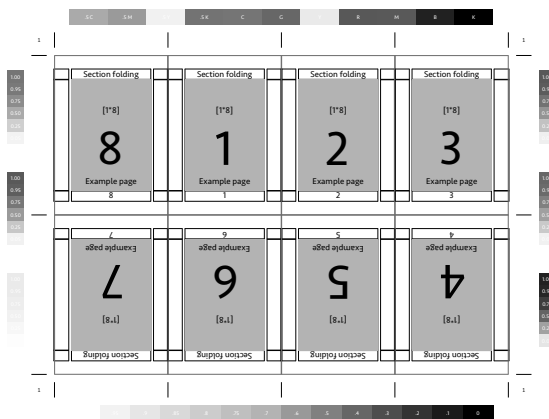


Figure 5 1*8

5 Flyers

Flyers are widely spread information leaflets often used for promotional purposes or as attractive give-aways for many occasions. The distribution is not yet providing these folding schemes, but they will be added in due time. The range will cover the most common flyer-types.

Flyer types

- Tryptichon this flyer has 3 pages recto and 3 pages verso
- Double window this flyer has 4 pages recto and 4 pages verso
- Z-folding flyers flyers with 8, 10 or 12 pages
- Map-flyer (12 pages) this flyer has 6 pages on recto an 6 pages on verso. It is a Z-fold flyer with a final cross fold.

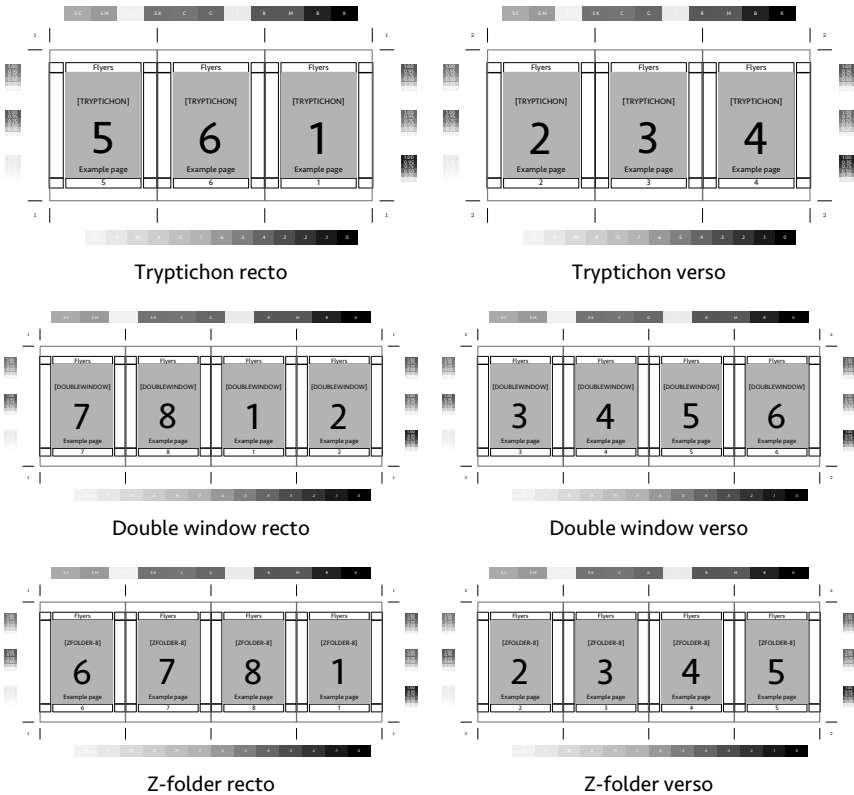


Figure 6 Flyers

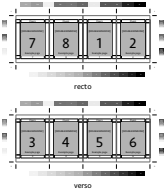


Figure 2 A double-window type flyer

Here then the setups used for this example:

```

\setpapersize[uspaper]{8.5cm*11cm}
\setmainfont{serif}
\settextfont{serif}
\setfontfamily{serif}
\setfontseries{normal}
\setfontstyle{normal}
\setfontcolor{black}
\setfontsize{12pt}
\setfontweight{normal}
\setfontslant{normal}
\setfontshape{normal}
\setfontscript{normal}
\setfontfeatures{normal}
\setfontoptions{normal}
\setfontlimits{normal}
\setfontkerning{normal}
\setfontligatures{normal}
\setfontmicrofilm{normal}
\setfontmultimedia{normal}
\setfontoutput{normal}
\setfontpath{normal}
\setfontsource{normal}
\setfonttarget{normal}
\setfontunits{normal}
\setfontwidth{normal}
\setfontxheight{normal}
\setfontyheight{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}

```

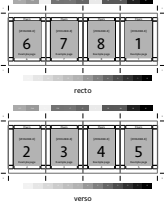


Figure 3 Example Z-folding flyer

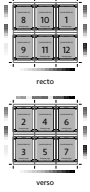


Figure 4 Map flyer



CONTEXT
Produces
Flyers!

September 2010, Brijlorg

"Context produces Flyers" is an example of how to create a flyer.

Typeset with ConTeXt-MkIV
Font : serif and 12pt
Paper : Colibri color printing paper 100gr
Foto : Frans Goddijn, Amsterdam
Design and idea
W. Egger, BOEDE, Sambek, The Netherlands, 2010



recto

Introduction
ConTeXt is the TeX environment which is developed most rapidly. It could be called a moving target. Recently the LuaTeX team implemented the Lua interpreter into TeX. This enabled Frans Rieger, the developer of ConTeXt to re-implement most of the ConTeXt code using currently more than 50% Lua-code instead of TeX code. A lot has changed indeed and before a stable version will be available it will yet take some time. However this does not mean that ConTeXt-MkIV does not work properly, only that certain things will need time to settle in. - Another important development is the rework of METAFONT into a library. This has boosted compilation times enormously because the library can be called at runtime and there is no need to call METAFONT as an external programme.
ConTeXt goes further than producing single (double) sided documents. It provides a mechanism, that allows the user to arrange pages on a larger sheet of paper for printing. This mechanism can be used for the production of book-sections as well as producing multipage flyers.

Multipage flyers
Strutting along the displays for flyers at the tourist office shows how many different ways exist to prepare a multipage flyer. At this moment ConTeXt offers six models of a multipage flyer.
For a page flyer the tryptichon is a suitable candidate. This implies the use of a landscape paper which is divided into 3 parts. By printing on recto and verso side you receive 6 pages. The production of such a flyer requires 2 parallel folds.
A similar approach is the "double-window" flyer. This flyer has eight pages, four on each side. The flyer is folded with three parallel folds. First the outermost pages are folded to the middle and finally the folder is closed by folding it in half again.
You can also produce a flyer with two times four/five/six pages with 2-folding (harmonica like). These folding schemes are provided as well. The folding is performed by folding the paper three/four/five times parallel one time forward and one time backward in turn. A 2-folding can also be folded in such a way, that the flyer is "closed". For this type of folding only page numbers higher than two are folded as a harmonica i.e.

page four is folded onto page three. Page one and two are folded around the pile to "close" the flyer.
There is also a map-type flyer. This flyer carries two times 6 pages. The pages are arranged in two rows of three pages. The flyer is folded first with a Z-folding and finally folded in half perpendicular to the first fold.

Coding of flyers
For the production of a flyer you need to define two paper sizes, one for the paper for printing and the second for the pages.
A standard flyer is about 21 cm high and the pages are ca. 10 cm wide.

For this example the paper sheet to be used for printing is 420 mm wide and 297 mm high. The grain direction is parallel to the short edge (i.e. a standard A1).
For the page size is defined according to the folding scheme. So you will have to calculate the page height and width from the base paper.
For a tryptichon:

```

\setpapersize[uspaper]{8.5cm*11cm}
\setmainfont{serif}
\settextfont{serif}
\setfontfamily{serif}
\setfontseries{normal}
\setfontstyle{normal}
\setfontcolor{black}
\setfontsize{12pt}
\setfontweight{normal}
\setfontslant{normal}
\setfontshape{normal}
\setfontscript{normal}
\setfontfeatures{normal}
\setfontoptions{normal}
\setfontlimits{normal}
\setfontkerning{normal}
\setfontligatures{normal}
\setfontmicrofilm{normal}
\setfontmultimedia{normal}
\setfontoutput{normal}
\setfontpath{normal}
\setfontsource{normal}
\setfonttarget{normal}
\setfontunits{normal}
\setfontwidth{normal}
\setfontxheight{normal}
\setfontyheight{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}

```

In order to arrange later on the pages on the defined paper, the defined page is linked to the paper size with
In the next stage you will need to provide arrangements for the page layout. This is achieved with the command
e.g. this flyer is prepared with the following options:

Before we start filling the pages one might want to setup the main language, make arrangements for the fonts to be used and define other components needed for the layout.
The last command we need is

The arranging command for the six types of flyers looks like follows:

```

\setpapersize[uspaper]{8.5cm*11cm}
\setmainfont{serif}
\settextfont{serif}
\setfontfamily{serif}
\setfontseries{normal}
\setfontstyle{normal}
\setfontcolor{black}
\setfontsize{12pt}
\setfontweight{normal}
\setfontslant{normal}
\setfontshape{normal}
\setfontscript{normal}
\setfontfeatures{normal}
\setfontoptions{normal}
\setfontlimits{normal}
\setfontkerning{normal}
\setfontligatures{normal}
\setfontmicrofilm{normal}
\setfontmultimedia{normal}
\setfontoutput{normal}
\setfontpath{normal}
\setfontsource{normal}
\setfonttarget{normal}
\setfontunits{normal}
\setfontwidth{normal}
\setfontxheight{normal}
\setfontyheight{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}
\setfontzheight{normal}
\setfontzwidth{normal}

```

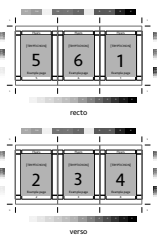


Figure 1 A tryptichon type flyer

verso

Figure 7 Map-folder: CONTEXT-flyer

6 How does page arrangement work?

Setting up the system

ConTeXt uses a system, where a page size is mapped onto a paper size. Both are defined as paper sizes. There are many commonly used paper sizes predefined, however it is no problem to define a custom size paper.

To get started, one needs 2 “paper” sizes.

A paper size for printing

A paper size for the pages

First we need to set up the paper and then we map the pagesize paper onto the print size paper.

```
\definepapersize[Printpaper][width=42cm, height=28cm]
\definepapersize[Arrangepaper][width=10.5cm, height=21cm]
```

```
\setuppapersize[Arrangepaper][Printpaper]
```

```
\setuplayout
```

```
[topspace=.5cm, backspace=.5cm,
```

```
header=0pt, footer=0pt,
```

```
height=middle, width=middle,
```

```
marking=color]
```

Once the setup is ready, one can apply the desired arranging scheme.

```
\setuparranging[...]
```

Compiling

There are two ways to prepare a document with page arrangement. It is possible to let ConTeXt take care of everything. To do this one issues the following command:

```
texexec or context -arrange yourfile.tex
```

The `-arrange` option tells ConTeXt to compile the document first without applying arranging. When it is ready it does one additional run for the page arrangement.

The other approach is to compile the document with the `setuparranging` command commented out.

The result is a document consisting of single pages. This is a good output for last checks.

```
texexec or context yourfile
```

If the result is ok, then a single run is performed with the `setuparranging` command uncommented.

```
texexec or context -once yourfile
```

It is important to keep in mind, that correct output is only obtained if the arranging run is performed a single time. Otherwise information contained in auxiliary files is overwritten, corrupted or lost.

7 Future work

For the preparation of flyers, a structure will be made available either based on Wolfgang Schusters module, which can be set up with variables, or the same approach which is used for all other page arranging schemes. Beyond the already present set of arranging schemes, those which are presented in this article and more will be added to the distribution.

Basically it is not that big an issue to add another arranging scheme. If you have a wish for a new arranging scheme it is always worthwhile to ask on the ConTeXt mailinglist.

Guide T_EX It: Uneasy Beginnings of Typesetters from the Perspective of Non-Typesetters

Nech T_EX nás být: pohled na krušné začátky sazečské z pohledu nesazečů

LIBOR SARGA

Abstract: The article describes the process of typesetting a proceedings in T_EX from the perspective of prospective typesetters along with challenges and obstacles encountered and solved during the work. Focused on the problems of generating a desired Table of Contents and captions of graphic objects, it further lists minor annoyances and tricks used to solve them. Also described is a field-proven electronic content management and synchronization system for different file versions utilized while working on the project in a decentralized fashion.

Key words: typesetting, T_EX, tocloft, caption, tables, figures

Abstrakt: Článek popisuje tvorbu sborníku příspěvků v systému L^AT_EX z pohledu amatérských sazečů spolu s problémy a těžkostmi, které byly při této práci zaznamenány a vyřešeny. Zvláštní pozornost je věnována peripetiím při tvorbě obsahu a popiskům grafických objektů. Nastíněny jsou také drobné problémy a triky, které je vyřešily. Rovněž je popsán v praxi vyzkoušený systém správy a synchronizace verzí souborů, použitý při decentralizované manipulaci s hlavní šablonou sborníku.

Klíčová slova: sazba, T_EX, tocloft, popisky, tabulky, obrázky

Introduction

Before we delve into how we end up typesetting a proceedings and what we went through while doing so, it is necessary to flesh out my prior experience with T_EX as this serves an important point of providing a background. Albeit short, the section is intended to provide information regarding my initial level of knowledge.

In September 2010 I began my first year as a doctoral student at the Department of Statistics and Quantitative Methods, Faculty of Management and Economics, Tomas Bata University in Zlín. I was aware even then that for producing quality scientific output it would be viable to choose a more sophisticated

The paper was partially supported by the ESF project No. CZ.1.07/2.2.00/07.0361.

system than widely-available commercial software residing at the university's computers. I started looking and very soon bumped into T_EX. I downloaded *The Not So Short Introduction to L^AT_EX 2_ε*¹ and promised myself to at least peek at it in the near future.

This I did and thus came to know things such as preamble, what to write in there, how to load packages and lots of other useful bits and pieces of what constitutes the system, which I had unfortunately no means of practically testing due to lack of infrastructure. That changed in early October when necessary equipment, including a fresh install of T_EX Live 2010 became a reality. I was pleasantly surprised that Lubor Homolka, another first-year doctoral worker at the department, and Pavel Stríž were both into T_EX as well and have often provided me with numerous theoretical and practical examples. I was then and am still now nothing but a beginner who decided to slowly familiarize himself with the 'tex' (phonetic transcript of my pronunciation at that time) using T_EXMaker editor. When I compiled my first document, I was elated:

```
\documentclass[a4paper]{article}
\begin{document}
Hello world!
\end{document}
```

I was pleased with myself, experimented as instructions told me, loaded packages, inserted pictures, switched languages. Lubor entered the realm of math typesetting. I was looking, jealous, as he compiled his dozens of lines long documents without any error while I explored aesthetically pleasing red messages the editor threw at me with steel precision and determination. Today as well as then I'm waiting, palpably discomforted, if the compilation, starting with innocuous

```
Process started
will end in victory
Process exited normally.
or if I get a dreaded
Process exited with error(s).
```

We saw our part of the last message at the time of typesetting, often more so than our families, friends, or beer kegs.

Proceedings

A relatively peaceful situation quickly gave way to something else after an e-mail came from Pavel Stríž, asking whether we would be interested in typesetting a proceedings in care of his brother's printing house². This sign of trust had me

¹<http://www.ctan.org/tex-archive/info/lshort/english/lshort.pdf>

²<http://www.striz.cz/>

excited: finally I will be able to show my extensive knowledge of PDF crafting, hampered by a mistake here and there, to the whole wide world. However, I was not completely able to filter out a notion of haste in the e-mail's text. Evidence, such as 'urgent work' and 'two or three weeks tops' left no doubt the process need have to be a quick one. We were further told the proceedings comprises of approximately 250 pages, 60 figures and 50 tables. My inner voice, warning me of possible repercussions, was quickly hushed and the deal was struck shortly before the middle of October. Pavel Stríž promised us backing in case we needed a hand which we decided to use only if no other option was evident to us. Out of the sample we were given the main issues were identified to be with the tables which would probably be necessary to convert into a unified format; and pictures, requiring conversion to PDF (we were yet to find out how).

On October 15, 2010 the proceedings in the .doc file was acquired. At 3.822 MB it contained all the articles in English along with figures and tables. We began to work. Since that day a frequently used communication channel was established between our side and Pavel Stríž's brother, Martin. He provided us with multitude of advice on basic configuration, a type of binding to be used and other mantras we have never previously heard of (we couldn't have since we had not been in the printing business, we concluded). The book's format was to be A5 (148 × 210 mm) as reflected in the settings of a template. We also had to admit weakness and reject making the cover using PStricks, a task too complex for either of us.

The first thing was to change the PDF viewer since Acrobat Reader 9 was not suitable for our purposes. Every PDF file the program opens is prevented from being changed, including with the editor commands. I found out later this is due to it not supporting DDE (Dynamic Data Exchange), an omission in many other viewers as well (Foxit PDF Reader). A solution is to create a simple batch file containing instructions to first terminate the process and then re-open it (a type of indirect data exchange) and link to this script instead to the viewer's absolute path. We did not know it then and decided to use Sumatra PDF that filled the gap.

Then, the settings of the editors were synchronized, especially font encoding (UTF-8) because Lubor decided against using commercial operating systems in favor of Linux, and these two different environments could cause problems. Also without UTF-8 in place comments in Czech became a gibberish of question marks and other exotic characters not put there previously by either of us even on the same versions of Windows. The `inputenc` package with `[utf8]` options provided a solution. Cropping the images was to be done by using a licensed version of Adobe Acrobat 8 Professional, listing the functionality in its features. An alternative is to use an internal command `viewport=` with the parameters being coordinates, allowing for automation of the process. We did not go this way as the thought of such declarations quickly sapped the color out of our cheeks.

The first step of image extraction was to convert the document into PDF, then using an open source tool PDF Split and Merge (<http://www.pdfsam.org/>) to extract the pages with the figures and finally unleash Adobe Acrobat Professional to crop them to external files. An alternative to the first step is to use pdftk (<http://www.pdfabs.com/tools/pdftk-the-pdf-toolkit/>). This ensured scalability unachievable by formats such as JPEG and PNG as per our own experience. Moreover, JPEG format is lossy by nature – not counting in the JPEG2000 variant which supports lossless compression – and suffers from generation loss, i.e. increasing of noise (specifically noise to information ratio) which is easily detectable visually when zoomed in. PNG rectified the problem, but PDF provided the ultimate scalability. However, as Martin poignantly quipped, ‘the cost of the software is higher than the average salary in the Czech Republic’. The program was the only commercial software used during typesetting, however (apart from operating systems).

We discovered a ‘Vlna’ macro³ by Petr Olšák which seeks solitary conjunctions and prepositions ‘a, i, o, k’ placed at the end of the lines (a mistake by all means) and links them with the following words. The command

```
vlna -l -m -n dokument.tex
```

assures automation of inserting such tildes (‘vlnka’ in Czech). The `-l` switch enables the L^AT_EX mode, `-m` ignores the math environment and `-n` does the same for the verbatim environment.

The macro is intended for Czech language text only, depressing us as the proceedings was in English. So we put emphasis on looking for numbers and units divided on two lines and placed several tildes in the early stages of our work. It became clear only later that any modification in the text renders many of them unusable and unneeded. A conclusion was that it is not advisable to solve all problems at once but instead create a hierarchy.

Trials and Tribulations

Since October the uphill struggle with the system has begun. One point became clear early on: the way to operate (that is, the way generating the least amount of errors) is to create a main template, inserting chapters by invoking `\input{*}` which produces much more clear and concise code structure. As for the packages, there were two of note: `tocloft` a `caption`.

³<http://ftp.linux.cz/pub/tex/local/cstug/olsak/vlna/>

Lists: tocloft

tocloft⁴ is a widely known and used set of macros for lists generation. Of those we needed three: chapters (ToC – Table of Contents), figures (LoF – List of Figures) and tables (LoT – List of Tables). The last two gave us our first serious headache. When we changed caption labels (‘Figure’ from ‘Obrázek’ and ‘Table’ from ‘Tabulka’) by means of altering the `caption`, the lists were subsequently populated only by their respective numbers, not the words themselves. We attempted to counter this behavior by leaving the default labels and redefining them globally by

```
\renewcommand*{\figurename}{Figure}
\renewcommand*{\tablename}{Table}
```

This produced the same result. I am not sure whether the problem is solvable only by changing the `babel` option to `[english]` instead of `[czech]` but I sincerely hope we tried that, too. Finally, we had to use a dirty trick. First, we indented all the items in the list by

```
\addtolength{\cftfignumwidth}{25pt}
\addtolength{\cfttabnumwidth}{20pt}
```

and then inserted the words ‘Figure’ and ‘Table’ to the existing space permanently by

```
\renewcommand{\cftfigpresnum}{Figure }
\renewcommand{\cfttabpresnum}{Table }
```

By fine tuning the values, we obtained the solution as seen in Figure 1.

<p>Table of Contents</p> <p>List of Figures 5</p> <p>List of Tables 8</p> <p>Introduction 10</p> <p>I. Global Economic Crisis Explored</p> <p>1. Core issues highlighted in the recent Revision to the Growth Diagnostic 16</p> <p>2. Economic Reform and Daring the Financial and Economic Crisis 20</p> <p>3. Growth Crisis in the EU 40</p> <p>4. Asset Price Fluctuations and the Financial Crisis 63</p> <p>II. Global Economic Crisis Outside of Eurozone</p> <p>5. The Effects of the Global Crisis on Turkish Economy and Existing Fiscal Policies for this Crisis 118</p> <p>6. Determinants and Absorption of Exchange-Market Pressure in Selected New EU Members 140</p> <p>7. Impact of the World Economic Crisis upon Measures of Convergence and Preparedness of the Candidate Countries to Join the Eurozone: Are We Better Prepared for the Euro? 168</p> <p>III. Impact of Crisis across the Economic Landscape</p> <p>8. Innovation to Irresponsible Behavior and Present Crisis 184</p> <p>9. The Influence of Official Development Assistance on Economic Development of the Selected Groups of Developing Countries around the World 210</p> <p>10. Evaluation of the Development of Unemployment Rate with regard to the Real GDP Growth Rate 245</p> <p>IV. In a Line of Conclusion</p> <p>11. The Information in the Economic Database and Analysis (Some clarifications about the role and uses of information) 254</p> <p>4</p>	<p>List of Figures</p> <p>Figure 1 Three Possible Scenarios for the Development of Potential Product in the Eurozone 44</p> <p>Figure 2 Average yearly inflation in Eurozone countries, 2002–2008 46</p> <p>Figure 3 GDP development in Eurozone countries before and after the introduction of the single currency Euro 51</p> <p>Figure 4 Development in long-term interest rates in selected Eurozone countries (1999–2007) 52</p> <p>Figure 5 Growth in Government Default Risk (the interest rate on CDF contracts for government bonds in selected Eurozone countries) 56</p> <p>Figure 6 Potential growth in the country groups of the EU 60</p> <p>Figure 7 Contribution of the capital accumulation to the potential growth 71</p> <p>Figure 8 Contribution of the TFP to the potential growth 74</p> <p>Figure 9 Potential growth in the new EMU (nominal) since 90' 75</p> <p>Figure 10 Potential growth in the EU Member States 75</p> <p>Figure 11 Potential GDP growth under different elastic demand growth rate 79</p> <p>Figure 12 Three defined paths of asset prices 83</p> <p>Figure 13 The movements of the Dollar/Euro exchange rate and technical trading signals, 1999–2008 89</p> <p>Figure 14 Technical trading signals based on intraday Dollar/Euro exchange rates, June 13, 2003, Summer data 90</p> <p>Figure 15 Technical trading signals for the S&P500 future contract, July and August, 2007 96</p> <p>Figure 16 Technical trading signals for WTI crude oil future contract, 2007–2008 96</p> <p>Figure 17 Appropriate trading signals of 1002 technical models and the dynamics of oil future prices, January 2007 to June 2008 99</p> <p>Figure 18 Dollar €/ exchange rate and purchasing power parity 103</p> <p>Figure 19 Dollar exchange rate and oil price fluctuations 103</p> <p>Figure 20 World market for crude oil, oil futures trading and oil price movements 104</p> <p>Figure 21 Dynamics of commodity futures prices and derivatives trading activities, 2007–2008 104</p> <p>Figure 22 Stock market value and net worth of non-financial corporations 105</p> <p>5</p>	<p>List of Tables</p> <p>Table 1 The depth of the two crises: Ten industrialized countries 24</p> <p>Table 2 GDP development in main regions 24</p> <p>Table 3 Comparison of the two crises: industrialized vs. non-industrialized countries: Growth of real GDP 29</p> <p>Table 4 Comparison of the two crises: New Member Countries and Neighbors: Growth of real GDP 30</p> <p>Table 5 Macroeconomic performance of the EU-10 31</p> <p>Table 6 External and fiscal balances in the EU-10 32</p> <p>Table 7 Main Macroeconomic Indicators: Comparison 42</p> <p>Table 8 Development of inflation and interest rates: comparison 46</p> <p>Table 9 Public institutions and debt servicing costs as a % of GDP 53</p> <p>Table 10 Selected indicators on the rate and impact of the economic crisis in EU-27 Member States 55</p> <p>Table 11 Potential GDP growth rate (annual average as percentage) 61</p> <p>Table 12 Labour productivity (annual average growth rate as percentage) 62</p> <p>Table 13 Potential growth in the European Union 67</p> <p>Table 14 Potential growth, current account and the investment ratio in the country groups 70</p> <p>Table 15 Potential growth and its factors in the country groups 72</p> <p>Table 16 Potential growth in the EU, USA and Japan 77</p> <p>Table 17 Features of three hypothetical ‘worlds’ of financial markets 87</p> <p>Table 18 Runs of the \$/€ exchange rate 1999–2005, daily data 92</p> <p>Table 19 Non-random components in the duration of exchange rate runs: daily data 92</p> <p>Table 20 Non-random components in duration and slope of exchange rate runs: 30-minute data 94</p> <p>Table 21 Hypothetical transaction tax receipts in the global economy 2007 113</p> <p>Table 22 Environment of The Crisis 120</p> <p>Table 23 Results of F-tests 126</p> <p>Table 24 Results of Diebold-Mariano and Misses-Richard tests 156</p> <p>Table 25 Estimation of Model 3 157</p> <p>Table 26 Estimation of Model 4 162</p> <p>Table 27 Descriptive statistics of GDP growth in Eurozone members, CEE and Baltic countries 172</p> <p>6</p>
--	---	--

Figure 1: ToC and a sample of LoF and LoT.

⁴<http://www.ctan.org/tex-archive/macros/latex/contrib/tocloft/>

This was our first major milestone which, apart from the fact it was achieved by a former WYSIWYG⁵ users, was the point where we realized even basic operations require a great deal of work.

I also concluded the documentation the authors of the packages provide are the source of all the information one needs despite some of them being a bit too technically demanding, i.e. `tocloft`. That's why we made it a habit of leaving one person (me) in charge of all changes made to (`tocloft` a `caption`). Lubor volunteered for the equally important task of cropping the pictures that, although hardly technical, was all the more demanding as to patience and accuracy required.

Labels: captions

We approached the `caption`⁶ package by Axel Sommerfeldt with a certain uneasiness due to our previous dealings with English labels but at the end it proved to be straightforward and painless thanks to its easily comprehensible and descriptive documentation. We were able to completely get rid of `\renewcommand{*}{*}` and change the labels from Czech to English by using an elegant solution in the preamble:

```
\usepackage{caption}
\captionsetup{figurename=Figure,tablename=Table}
```

We strived to parametrize as much as possible by the packages' options rather than using global modifiers that could possibly lead to further, wholly unexpected problems.

Hyperlinks in captions made us equally worried. Before we utilized the `url` package each slash had to be preceded by a backslash, proving to be our bane as after we decided on `url` all the backslashes had to be removed again. Without using `url` the hyperlinks were further treated as plaintext and therefore unclickable in the PDF. Another conclusion abound: familiarize myself with the possibilities `TEX` offers before attempting to devise my own dirty solutions.

The resulting labels are shown in Figure 2:

To all graphic objects the string `[!hpb]` was appended, courtesy of Pavel Stríž who proposed it after our complaints that all pictures are positioned at the end of the text, putting a reader in a dire situation especially when the author repeatedly commented on the object within the text body. Later on, I found out the characters serve as a guide on where to put the objects in `TEX`'s notation.

My last conclusion came from seeing advantages of using `\textwidth` and `width` instead of `\scale`, a point we proved repeatedly. While the former scales the object in relation to the width of the text on a page, the latter does so absolutely. We had to very often tweak the `scale` manually as the object was

⁵What You See Is What You Get editor allows to see the changes made to the document without the need to re-compile it.

⁶<http://www.ctan.org/tex-archive/macros/latex/contrib/caption/>

Table 7 Main Macroeconomic Indicators: Comparison

Source: EC (2008)

Figure 41 ISE National 100 Index (Based on the Closing Prices)

Source: The Undersecretariat Turkish Treasury, Ekonomi Sunumu, http://www.hazine.gov.tr/irj/go/km/docs/documents/TreasuryWeb/Statistics/EconomicIndicators/egosterge/Sunumlar/Ekonomi_Sunumu_ENG.pdf, p. 129

Figure 2: An example of figure and table labels.

positioned outside the bounds of A5. By using any value lower or equal to one in the first declaration (for instance `\width=0.5\textwidth`) the problem is avoided.

Tables

Tables were another challenge we had to face when we finished with figures. There were two principal options in the matter: either convert all of them to \TeX or safely crop them as we did previously. Because my greatest achievement in this regard at that time was the `{c|c|c}` declaration, causing all the text in three columns to be centered, and because several tables looked like even A3 format was too small to accommodate them, the second way was definitely preferred.

The problems arose immediately, first of which was the output format. Several extremely large tables were divided on two consecutive pages and therefore needed to be joined before we were able to crop them at all. I could not achieve this on my non-widescreen monitor so I set the page format in Microsoft Word to an extremely large value (B0, or 1000×1414 mm was sure to work) instead, then using a virtual printer PDFCreator⁷ to set the same output format and printed to PDF.

Before I stumbled upon the correct value, though, some time has passed. In Adobe Acrobat, the resulting behemoth had to be scaled down for the lack of 1.4 meters wide monitor in our possession. We were proud... until we found out the tables were not converted to the same format which required us to do just that and then repeat the process of virtual printing and cropping again. Fortunately, Martin Stríž came to our help and diligently sit through the task, helping us when we realized the results of several days worth of work are useless.

Miscellaneous

During the course of the work file management and naming were a constant issue. We had several versions of files, tables, figures, templates, working and

⁷<http://sourceforge.net/projects/pdfcreator/>

to-be-discarded experiments in the e-mails' attachments, removable media and folders. Maintaining order was imperative especially because we were working in a decentralized fashion with most of us having different way of keeping order in our repositories. This resulted in some working experiments to be repeated because the template the code was re-written by another piece of working code elsewhere. What I finally came to do was to save a working template at the beginning of each day and after the work was done copying the additions to the backup file before compiling it. Thus we always had a working version as a core component, modified as needed. In case of procuring a file for control, the maximum offset would be 24 hours.

As for file naming convention, the spontaneous system we devised and tested consisted of `figXX` for figures, and `tabXX` for tables. When the table was divided into two pages, the files were named `tabXXa` and `tabXXb`, the joined table `tabXX` as usual. The notation for individual chapters was `kapXX` (meaning chapter), the main template was `sablona_hlavni` (meaning main_template). Even by quickly looking at the contents of the folder it was possible to make out the structure easily.

Several marginal challenges appeared, solved by diligently looking through forum boards of L^AT_EX users. A simple action such as putting a dot after chapter name still resides in my bookmarks. Answers to such questions, sought after by beginners, are to professionals a matter of invoking a corresponding declaration (`\renewcommand{*}{*}`) or loading a package containing the functionality (`titlesec`). We certainly forgot lots of things as we haven't used them since, and it was also possible to solve a lot of these problems more easily than we did back then. For instance, we loaded the `fancyhdr` package and used it only to produce a plain page with no numbering, a feature easily achieved by setting `\thispagestyle{empty}`. We handled clubs and widows⁸ by increasing the scale of the preceding picture by avoiding them while the correct solution is to set the correct value in `\widowpenalty=XX` and `\clubpenalty=XX`.

One of the things we did not pay attention to initially were quotation marks. We were not too experienced in this regard in Czech language and asked for the correct way of handling them. The answer in the form of warning was that they are problematic and that we had to make sure we had them right. We promised we would be careful, and did the mistake anyway. The situation was complicated by the fact the authors of the articles were foreigners (meaning the quotation marks were not 'Czech'), the proceedings was printed in the Czech Republic, though (suggesting Czech quotation marks should have preference). We chose the second argument to base our assumptions on. This caused us to sift through the whole text again and correcting that mistake. To avoid further complications

⁸In typography, a club is the first line of a paragraph positioned on the last line of the page; a widow is the last line of a paragraph positioned on the first line of the page.

when searching according to Alt+ codes, we copied the quotation marks off Czech and English version of Wikipedia. Czech quotation marks look like „this“, English like “this”, or to put it simply, Czech quotation marks resemble the pair 99 down and 66 up while the English ones match 66 and 99 up.

However, we soon discovered a catch in the form of inserted quotation marks which also exhibits this peculiar property. Czech inserted quotation marks „look like ‚this‘“, the English ones “look ‘like this’”, however (so, in terms of pairs, 6 down 9 up for Czech language, 6 and 9 up for English). New sifting through the text and assuring all inserted quotation marks are the English version. After this was done we were confident we got through the worst of it. It came to us as a surprise when we found out some authors did not respect typographic conventions and instead of standard characters as seen above used the apostrophe (') with nearly unrecognizable difference in appearance in the source text. Third sifting and close examination of the English keyboard layout in search of another characters possibly representing quotation marks bore no fruit and we were confident this was the last time we heard of any problem concerning quotation marks. Only later we found there was another ‘type’ of inserted quotation marks in form of acute. The final sifting was done in a matter of minutes.

Conclusion

Typesetting the proceedings greatly enriched us both. With the help of Pavel Stríž and his brother Martin the work was finalized before the deadline. After proofreading, additional corrections and production of the cover by a third party we received a message announcing the book is finished. When we got the physical copy we were rightfully proud as every equation, table, figure, quotation mark, punctuation mark and dot was revised multiple times. Even though several mistakes came to light before the printing we were not having too ambitious a goal of leaving no mistakes behind but rather to find out if we can do it.

We did and learned a lot of things in the process we now use when preparing presentations for the students whom we also try to enlighten as to the ways of open source typography, particularly $\text{T}_{\text{E}}\text{X}$. They should be allowed to see that it is possible to typeset not only using commercial software but with free solutions as well. Taking into account further tools for statistics, such as programming language R, gretl or gnuplot graphic utility, we may safely conclude quality science can be done with minimal (zero) costs involved.

sarga@fame.utb.cz
Faculty of Management and Economics
Tomas Bata University in Zlín
Mostní 5139, 760 01 Zlín

Typesetting of Tables and Lists and Other New Features in T_EXonWeb

Sazba tabulek, seznamů a další novinky v T_EXonWeb

JAN PŘICHYSTAL

Abstract: This article describes new features in T_EXonWeb. T_EXonWeb is a web application that allows us to use T_EX/L^AT_EX typesetting system without needing its installation on a local computer. One of the most important characteristics of this application is to help beginners to start working with T_EX/L^AT_EX. It offers them a lot of tools such as table and list wizards to ease their first steps.

Keywords: T_EXonWeb, T_EX/L^AT_EX

Abstrakt: Tento článek popisuje nové vlastnosti systému T_EXonWeb. Jedná se o webovou aplikaci, která umožňuje využívat typografický systém T_EX/L^AT_EX bez nutnosti instalace na lokální počítač. Jeden z nejdůležitějších rysů této aplikace je zaměření na pomoc začátečnickům pracujícím s T_EX/L^AT_EX. Nabízí jim mnoho nástrojů pro usnadnění prvních kroků, jako například návrhář tabulek a seznamů.

Klíčová slova: T_EXonWeb, T_EX/L^AT_EX

Introduction

As we mentioned in [1], one of the main aims of T_EXonWeb application is to provide a simple interface for document processing using the typographic system T_EX/L^AT_EX. Many users would like to produce high-quality documents but are not familiar with the non-trivial T_EX/L^AT_EX installation and configuration. Now T_EXonWeb could offer them more than just a simple text editor. Beginning users often fail in making more complicated parts of their documents. Our experience shows us users have lot of problems with typesetting tables, lists or including graphics into their documents. This was the reason why we decided to implement wizards with a simple interface helping beginners to produce T_EX/L^AT_EX code of these more complicated document parts.

Tables

$\text{T}_{\text{E}}\text{X}$ onWeb application offers beginners a toolbar for interactive insertion of $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ commands. For example any user could easily select a text part with his/her mouse and click the button to change the text style to italics or bold font and the editor inserts the right $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ command. But there are also more complicated parts in documents. One of them could be the table. Inserting the source code of it could not be done by just one mouse click. This is the time for source code wizards.

Table Wizard in $\text{T}_{\text{E}}\text{X}$ onWeb is based on $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ `tabular` environment and consists of two windows. In the first one, the user specifies the table size — number of rows and columns. In the second window appearing after the size specification, there is a table designing area similar to the style in which users design tables in text processors or spreadsheet applications. In this area the user can fill a text or numbers to cells just by clicking into them. The size of the table specified in the previous window is not unchangeable. The user can add new or delete unused rows or columns by clicking plus (+) symbol in the table designer header or clicking icons in *Edit & Alignment* section. This section is available after selecting the whole row or column by its header. Values in cells could be aligned by selecting the whole column and specifying the style (left, center, right, justify).

Another important attribute of tables are their borders. The user can set the border style just for the cell, for the column, row or for the whole table (clicking the button in the upper left table header). There is a set of icons illustrating available border styles. Border line styles could be set to none, single or double.

Cell merging is also available. First it is necessary to select one cell. Then holding Shift key and moving cursors the user can select more cells. After it clicking the icon in *Edit* section merges cells together. *Table Wizard* does it by `multicolumn` and `multirow`¹ macros.

The table could be placed into $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ floating environment `table` by checking the box *Table environment* and filling its properties (position and title).

The table source code could be sometimes quite complicated. Therefore users have the possibility to view it by *Preview* button before inserting it to their own documents. Button *Insert* places the source code to the document and *Close* button closes the table wizard. We recommend not to close the wizard before ensuring the table looks like we want to. The window is not modal, which means, the user could insert the table code and if not satisfied he could delete it in his document and continue designing the table. When the wizard window is closed it is not possible to continue designing in the table wizard. The user should continue just in editor or start again.

¹provided by `multirow` package

Lists of items

Another document part making beginners problems is the list of items. The T_EXonWeb offers *List Wizard* helping users to design ordered, unordered or combined multilevel lists.

First it is necessary to specify the style (ordered/unordered) for the highest list level. First item of the list is visible thus the user can type a text there. The next item appears after clicking the icon with plus (+) symbol or just pressing Enter key. Deleting of the item is also simple, just click the red cross icon.

If there are more list items, the user can modify their order in the list by up/down arrows. This means to move the second item to the first position etc.

Left/right arrows modify the item level in the list. The user can move selected item to the lower level clicking the right arrow or move the item to the upper level clicking the left arrow. Each list level could have a different style (ordered/unordered).

Preview button shows the source code before inserting it to the document, *Insert* button places the source code to the document and *Close* button closes the *List Wizard*. There is a similar limitation as we discussed in the tables section. This means after closing the window it is not possible to continue designing the list in wizard.

Graphics

Because T_EXonWeb is a web application running on a distant server, users must upload graphics to their accounts there to see them in documents. Sometimes it makes problems to beginners and also setting properties of a picture could be complicated. This process is simplified by *Insert Picture wizard*.

The user first distinguishes if he wants to place already uploaded picture to his document or include a picture from his local computer. If he selects to *Upload*, there appears an upload form where he can find the picture within his computer. After selecting *Existing* there appears a list of pictures already uploaded to his account. Then the user can specify if to place the picture to L^AT_EX floating environment **figure** and define its position and title.

Properties section enables to specify the width, height and rotation angle of graphics. *Insert* button places the source code to the document and if necessary uploads the picture to the user account on the server.

Other tools

Besides these wizards mentioned above there are few other things helping beginners to create high-quality documents in T_EX/L^AT_EX. The first one is a set of

predefined document templates. There is e. g. a template for a curriculum vitae or letter. The set is still being extended with new templates.

The last very useful tool is *Spell Checker*. This is based on `aspell` application and enables the user to build his own dictionary to avoid highlighting special words. Spell checker highlights misspelled words in the text with yellow color and the active word with red color. It also displays a window with the list of alternatives for the misspelled word where the user can find the right word and replace it with button *Replace*. If the word is right but not in the dictionary the user can add it using button *Add*. He could also ignore the word or skip to the next/previous word. The selection of misspelled words is also possible by mouse. Just click the highlighted word.

Conclusion

TeXonWeb is not intended as a full substitution of specialized T_EX/L^AT_EX editors installed on local computers. Web page could not offer such comfort in writing documents. But in some situations it could be useful, especially for beginners who want to try how T_EX works and are not familiar with all L^AT_EX macros.

TeXonWeb is still being developed and new functions and options are added quite often. Now we work on a multilingual support, document templates, editor improvements and hard on a file manager to offer more user friendly management such as copying, renaming and uploading files or work with directories.

You can try TeXonWeb at the url <http://tex.mendelu.cz/en>.

References

- [1] PŘICHYSTAL, J. *TeXonWeb*. TUGBoat. [online]. TUGBoat. 2009. Vol. 30, No. 1, p. 18–19. (ISSN 0896-3207.) Available at: <http://www.tug.org/TUGboat/tb30-1/tb94prichystal.pdf>

Jan Přichystal
Department of Informatics, Faculty of Economy, Mendel University, Brno,
Zemědělská 1, 613 00 Brno, Czech Republic
jprich@pef.mendelu.cz

ConT_EXt for 'Zines

Elektronický časopis ConT_EXtem

TIMOTHY EYRE

Abstract: The article describes the design of the New Escapologist magazine, our motivations for using ConT_EXt, some of the typographical features of the magazine and my experiences with using the ConT_EXtMark II macro package.

Key words: electronic magazin, ConT_EXt, typesetting.

Abstrakt: Článek představuje sazbu elektronického časopisu New Escapologist. Jsou popsány důvody užití systému ConT_EXt, typografická úptava i vlastní zkušenosti autora s ConT_EXtem Mark II.

Klíčová slova: elektronický časopis, e-zin, ConT_EXt, sazba.

*tim (at) newescapologist (dot) co (dot) uk
London, England*

ConTEXt for 'Zines

Based on a poster presented by Tim Eyre at the Fourth International ConTEXt Meeting.

NEW ESCAPOLOGIST [1] is an Epicurean and mildly political amateur magazine edited by Robert Wringham of Montréal. Its general theme is escape from the daily monotony of office work by means of unconventional lifestyles such as minimalism and entrepreneurship. Issues appear approximately twice a year, with Issue Five nearing completion as I write.

Although I have written a few short articles for *New Escapologist*, my primary involvement is with the technical aspects of production. Specifically, I typeset the magazine and to do this I use ConTEXt. This article describes the design of the magazine, our motivations for using ConTEXt, some of the typographical features of the magazine and my experiences with using the ConTEXt Mark II macro package.

PAGE DESIGN

I met Robert Wringham on the Internet blogging site LiveJournal. One day he sent me a physical copy of the new magazine that he had conceived. This was the first issue of *New Escapologist* and Robert was interested in any feedback I had on

the publication. Robert was clearly serious about the project: he had endowed it with an ISSN and had solicited a foreword. I did not consider myself qualified to comment on the content. However, my long-standing passion for typography caused me to wince at the layout (see Figure 1). The articles had been prepared using Microsoft Word and were printed in 9pt Arial on A4 office paper. The pages were crudely side-stapled together. The presentation did not do justice to the concept or the content.

Robert agreed with my comments on the typography and asked me to take on the rôle of typographic advisor to *New Escapologist*. I agreed and shortly afterwards Robert sent me a valiant initial effort at typographical quality that he had painstakingly prepared in Microsoft Word. I quickly replied with an improved version of the layout, which TEX had enabled me to produce in a few minutes. This example of my work caused Robert to immediately promote me to the rôle of Chief Typographer.

The page design that we devised between us consists of a square page with two columns of

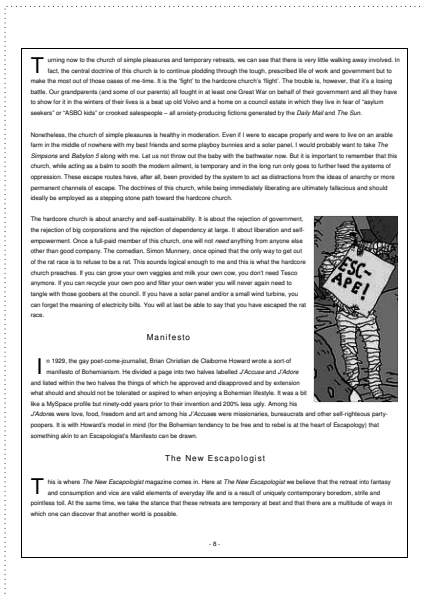


FIGURE 1: The weak typography of the original design for Issue One. Each page included a frame, perhaps to stop the words falling off.

ragged-right text and no hyphenation. We use a paper size of 51pc x 51pc, which is one of the two square sizes offered by the on-demand printing service Lulu. (*New Escapologist* uses Lulu for all its printing.) The main body of the text is set at 12pt with 19½pt of leading. These dimensions mean that a standard column holds 25 lines.

Robert selected the font Adobe Jenson, which

is the sort of font that Bringhurst classifies as 'Humanist' [2]. This font was, like Adobe Minion, designed by Robert Slimbach of Adobe Systems and is inspired by the Renaissance types of Nicolas Jenson and Ludovico degli Arrighi. Jenson is subtly eccentric in its design, which fits in well with the gently radical ethos of the magazine. To start with I considered using the sans-serif font Adobe Myriad for the headings, but in the end we decided to use Jenson only. The bold version of Jenson did make an appearance in Issue Two (the first one I typeset), but we did not use it in any subsequent issues. This includes Issue One, which we re-set in the new format and re-released after Issue Two, declaring the original A4 version to be apocryphal. Continuing the theme of typographical parsimony, I restrict myself to just three text sizes: \tf, \tc and a locally-defined \Tfe. This avoids the trappings of ransom-note typography and recognises the magazine's ethos of minimalism and simplicity.

An important factor in the design of *New Escapologist* was that it is not a money-making venture. As such, we were neither required to cram as much text as possible into a given space, nor were we required to pad out the text to create an illusion of value. Instead our design decisions could be made primarily on the basis of aesthetics.

This freedom is a great luxury for a typographer and one of the joys of being an amateur.

A further advantage of the non-commercial nature of *New Escapologist* is that it carries no advertising. This enables us to maintain a high level of design consistency throughout each issue, and indeed through successive issues. This design consistency now extends as far as the magazine's website and blog [1], which Robert had re-designed to echo the design of the magazine.

As Bringhurst writes in [2], historical manuscripts are a valuable resource for typographers to draw upon when designing layouts. As I developed Robert's initial layout, I used as inspiration an ancient Ge'ez liturgical text that I had seen in the Matenadaran Manuscript Museum (Մեսրոպ Մաշտոցի անվան հին ձեռագրերի ինստիտուտ) in Yerevan, Armenia. Indeed, well before I became involved with *New Escapologist*, I created a mock-up of this page layout using Xe_{La}TeX, complete with fidäl (ፊደል) script, as an exercise.

We typeset *New Escapologist* articles in two different flavours of layout: 'Articles' and 'Anecdotes'. Articles tend to be longer and are supported by a dedicated title page with an illustration (see Figure 2). Anecdotes tend

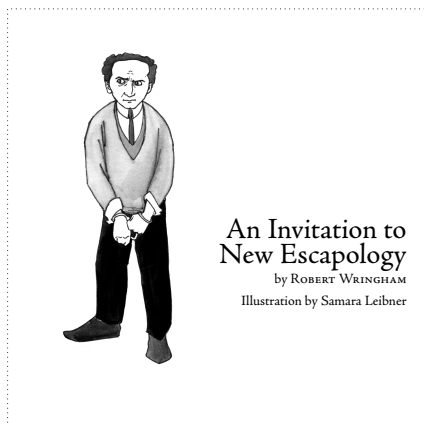


FIGURE 2: An Article title page.

to be shorter and have no separate title page (see Figure 3). The macros that I developed to produce these two layouts provide a sound framework that makes it easy to achieve consistency. I typeset the present article using the Anecdote layout adapted for A4 paper.

WHY CONTEXT?

The main reason that I chose ConT_EXt is simply that I am more experienced with ConT_EXt than I am with L^AT_EX and I couldn't face writing everything from scratch in plain T_EX. However, I understand that the emphasis of L^AT_EX is on the efficient production of technical papers

Escaping distractions

NATH SCOTT demonstrates how to achieve mindfulness.

YOU ARE sitting down. You hold a magazine in your hands. Your fingers crease the pages as you begin reading this article. As these words reach your retina, electrochemical explosions light up your brain and attempt to assign the words meaning. You breathe in. Your heart beats. You are alive, but you are not really living. You look, but you do not see.

Like most people today, you are a slave to distractions, incapable of engaging fully in your life without drifting off, worrying, or checking the Internet. This can't go on. For those who want to escape the banal elements of modern life, I have five steps that will enable you to actually experience your life, rather than merely watch it pass by.

1. ESTABLISH PRIORITIES. It's not that you can't see the wood for the trees; you can't see the trees for the leaves. You need to take several steps back to remind yourself of your priorities. You never stop to think whether what you're doing is of any

value. Secretly you worry that, if you did step back to see what it all amounts to, you'd realize that you'd wasted your life. You did what others expected you to do, rather than creating your own hierarchy of values that would allow you to make more informed decisions. But where to start?

You could begin by fantasizing about your death. What utopias would you want at your funeral? What virtues and achievements would you want people to praise? Because you are the one doing the imagining, you can make them as outrageous as you like. After all, you are the person that is going to have to live the life that will make those things possible. From this, you can begin to understand what you really value: be it kindness, intelligence, material success, creative fulfillment or whatever.

2. KILL COGNITIVE DISSONANCE. Now you have your priorities, you will find that it is much easier to make good decisions about what to do from moment to moment. If, however, you still find

41

Escapologist. To make the most efficient use of the limited space available I set the text fully justified and hyphenated and therefore was able to use the margin kerning and font scaling capabilities of pdfT_EX. I considered feeding the code I had developed through to the main magazine but decided against it.

The T_EX installation that I use is W₃₂T_EX, maintained by Akira Kakuto (角藤亮) [3].

The reason I use this is not a political rejection of T_EXLive but simply that several years ago W₃₂T_EX was the only T_EX installation I could find that made Japanese typesetting easy [4]. I shall eventually migrate to T_EXLive.

TYPOGRAPHICAL FEATURES

In this section I shall describe some of the salient typographical features of *New Escapologist*. None of these features is revolutionary but they do illustrate how ConT_EXt and X₃T_EX facilitate the creation of typographically pleasing material with indulgent flourishes.

Of initial interest are the drop capitals, or lettrines. For these I used a ConT_EXt module written by Taco Hoekwater [5]. I have found Taco's module to be robust and flexible, allowing fine control while still being simple to use and apparently unbreakable. Arguably we make

FIGURE 3: The start of an Anecdote.

whereas ConT_EXt is more orientated towards non-technical publications. Furthermore, I had already written a ConT_EXt-based format that I could use as a starting point for developing the macros for *New Escapologist*.

I chose to use X₃T_EX as the typesetting engine rather than pdfT_EX. The main reason for this was to make it easy to access Adobe Jenson as a TrueType font. I am a fan of the microtypographic features of pdfT_EX but these are less relevant in a ragged-right layout. However, Robert and I did once create an A4-format promotional pamphlet containing two sample articles from Issue Two of *New*

excessive use of drop capitals in *New Escapologist* as a result.

The most time-consuming typographical feature of *New Escapologist* is the use of pull quotes. Robert Wringham likes to use pull quotes in the interviews that he publishes in the magazine. For maximum impact, we place the pullquotes in the centre of the page and wrap the text in columns on either side around the quote (see Figure 4). Unfortunately I have found no shortcuts for doing this: I typeset the quote as a separate PDF file and then include it as an image in the article using the powerful layers feature of ConTeXt. I then use `\startnarrower` and `\stopnarrower` to wrap the text around the quote. This is painstaking work requiring numerous compiler runs and much trial and error. However, the results are worth the effort.

New Escapologist has only once published poetry; this was in Issue Two where Robert included two poems by Edward Lear. This provided me with an opportunity to make use of the more elaborate ligatures provided by the font Jenson: I made use of `ſt`, `ct` and Jenson’s ornate variant of `Q`: `Q̇`. While distracting in prose, these ligatures added a typographical flourish to the poetry. TeX macros provided a natural way to robustly implement the complex indenting



FIGURE 4: Text wrapped around a pullquote.

This is painstaking work.

structure of the stanzas in these poems.

Although the layout of *New Escapologist* has stabilized, I continue to refine the macros and add new features. I expect this process of refinement to continue. My most recent development was for Issue Five. Here I introduced spaced capitals for the headings, a feature I have also used in this article. I implemented this spacing by means of the ConTeXt macro `\spaced`, which makes basic tracking easy. However, I had to break out some typographical heavy artillery to cope with headings that themselves included macros.

Finally, the feature of the *New Escapologist*

macros that I have developed of which I am most proud is not related to typography at all. Instead it is related to ease of use. After typesetting two issues of *New Escapologist* I came to find the grunt work of basic formatting tedious and time-consuming. Robert was willing to take on this work but, as a relatively non-technical person, he was concerned about the difficulty of creating T_EX markup. I addressed this by re-working the code I had written into a well-defined set of clearly-documented macros and teaching Robert how to use them. Robert approached the world of T_EX with some trepidation at first but was pleasantly surprised at how easy it was to create articles. We have now settled into a productive routine where Robert does the bulk of the work as part of the editing process and I handle only the more technically challenging parts of creating an issue of *New Escapologist*. This is empowering for Robert and allows me to concentrate on the parts of the production process that interest me most. The fact that T_EX reliably produces the same output on different machines given the same source files is a major advantage here, as is the fact that T_EX and ConT_EXt are free (although the font Adobe Jenson is not). Robert is unlikely to have been enthusiastic about paying for a professional typesetting package.

USING CONTEXT

While working on *New Escapologist* the most notable thing I have found with ConT_EXt is that it has numerous powerful features built in and fully integrated with the rest of the macro package. First and foremost among these is the facility to typeset on a grid. Grid-based typesetting sits alongside small capitals and old-style numerals as a hallmark of quality typography. However, the mathematical heritage of T_EX means that typesetting on a grid does not come naturally to it. After spending hours tweaking the interline space in the first two issues of *New Escapologist* I decided that we should use grid-based typesetting and ConT_EXt made it delightfully easy to do so. The simple `\showgrid` command makes validating the grid simple and is also extremely helpful for widow control.

Another example of a powerful feature of ConT_EXt that ‘just works’ is layers. I have found layers can make what would otherwise be tricky typographical challenges relatively straightforward. They were especially useful for implementing pullquotes.

A major benefit I find to using ConT_EXt is the unified documentation set [6]. This combined with the ConT_EXt Garden website [7] and mailing list archives [8] means that I can usually

find the answers I need. However, sometimes the trail goes cold and my impatience causes me to use a workaround rather than raise a question on the mailing list.

The area where I had the most significant difficulties was with footnotes. When setting single-column text I have few problems with the footnote features in ConTEXt. However, after prolonged wrestling with the macros in the dual-column format of *New Escapologist* I eventually gave up and asked Robert to avoid footnotes where possible. I set the few footnotes that remain by placing the text by hand rather than using any footnote macros. Hans Hagen himself acknowledges footnotes as being a tricky area in [9]. A major difficulty with footnotes in a multi-column environment is establishing what a sensible and aesthetically pleasing layout should be.

I find font control in ConTEXt documents to be intuitive and easy, especially in *New Escapologist* where we use relatively few fonts. The difficult bit was setting up the typescript in the first place; I still find typescripts to be something of a black art when I make minor tweaks.

The complexities of the ConTEXt macros mean that multiple compiler passes are necessary for a single build. Furthermore, the columns feature

of ConTEXt seems to slow down the building process considerably. However, compiling only a few articles at a time is a perfectly acceptable workaround.

CONCLUSION

ConTEXt has been crucial in turning Robert Wringham's vision of *New Escapologist* into reality. It has provided a sound typesetting platform for our completely non-technical publication and, being free, it is within the budget of our non-profit amateur venture. The advanced typographical features of ConTEXt enable us to produce a 'zine with outstanding typography. Furthermore, the power of ConTEXt has enabled me to create macros that make production of new issues a process simple enough to be largely undertaken by the non-technical editor. I have encountered some technical niggles when using ConTEXt but overall it has proved to be a package that enables the creation of material of high typographic quality without compromising productivity.

REFERENCES

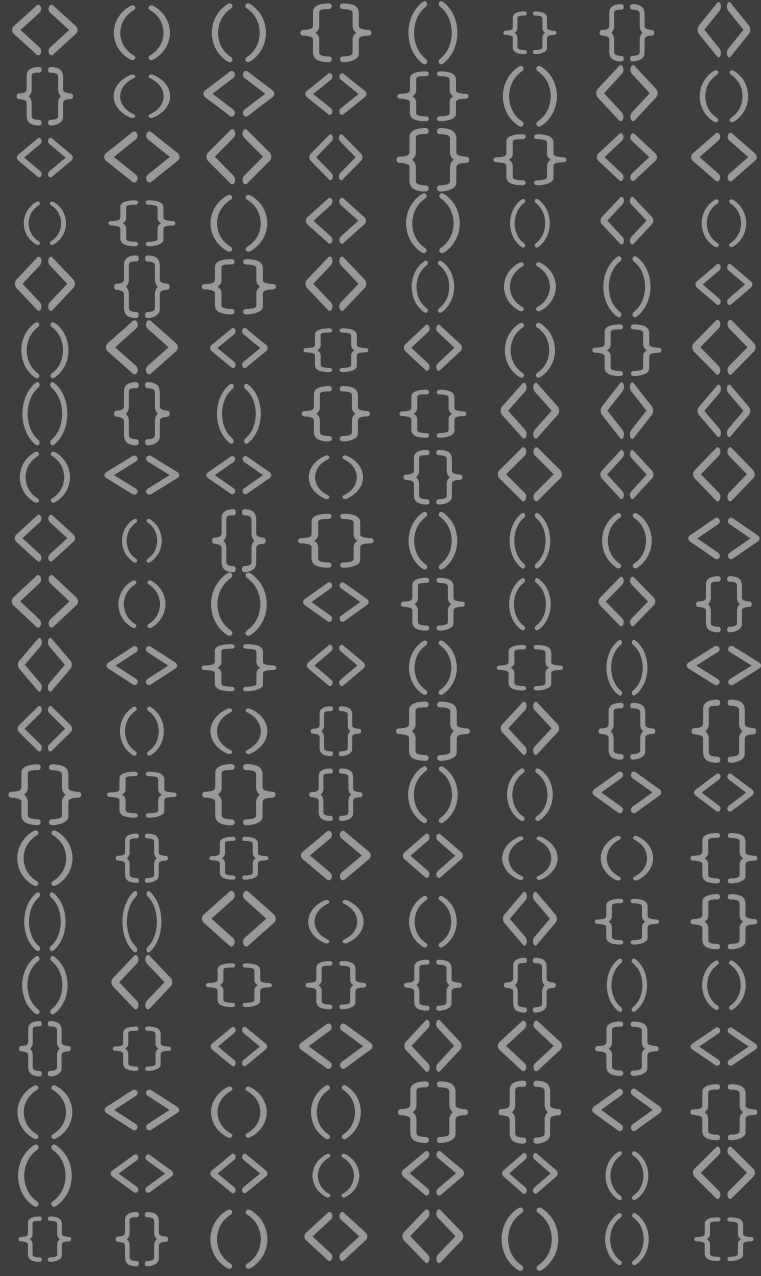
- [1] New Escapologist Issues 1–5, ISSN 1755-5671, available from <http://www.newescapologist.co.uk>
- [2] Robert Bringhurst: *The Elements of Typographic Style*, 2nd ed., 1996.
- [3] <http://www.w32tex.org>
- [4] Timothy Eyre: Typesetting Japanese with pT_EX. In *The Zpravodaj Journal*, 20(3), pp. 152–173, 2010. ISSN 1211-6661.
- [5] Taco Hoekwater: Lettrines for ConT_EXt. In *MAPS*, 32:26–28, 2005. <http://www.ntg.nl/maps/32/07.pdf>
- [6] <http://www.pragma-ade.com/overview.htm>
- [7] <http://www.contextgarden.net>
- [8] <http://www.ntg.nl/pipermail/ntg-context> and ntg-context@ntg.nl.
- [9] Hans Hagen: *ConT_EXt the Manual*, November 12 2001. <http://www.pragma-ade.com/general/manuals/cont-enp.pdf>

CONTACT

Email: tim@newescapologist.co.uk



FIGURE 5: The house staff of *New Escapologist*, as illustrated by Samara Leibner. I stand third from the right.



MkIV Hybrid Technology

MkIV Hybrid Technology

Hybridní technologie MkIV

HANS HAGEN

Abstract: The paper presents development, new features and tools of Lua_{T_EX} and Con_{T_EX}tMkIV.

Key words: Lua_{T_EX}, Con_{T_EX}tMkIV, Mark II, Mark IV.

Abstrakt: Příspěvek představuje rozvoj, nové vlastnosti a nástroje Lua_{T_EX}u a formátu Con_{T_EX}t Mark IV.

Klíčová slova: Lua_{T_EX}, Con_{T_EX}tMkIV, Mark II, Mark IV.

References

- [1] Lua_{T_EX}home page. Available at URL: <http://www.luatex.org/>
- [2] The Programming Language Lua. Home page.
Available at URL: <http://www.lua.org/>

pragma (at) wxs (dot) nl
PRAGMA ADE, Ridderstraat 27
8061GH Hasselt, The Netherlands

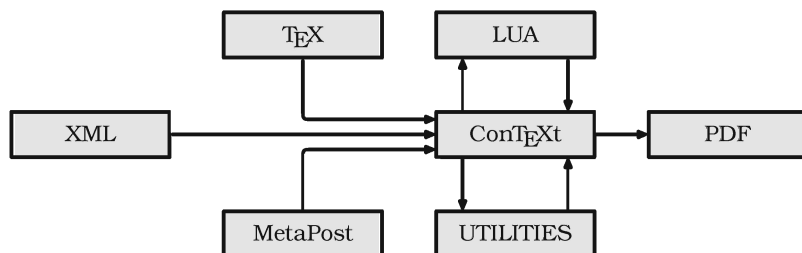
Introduction

We're halfway the development of Lua \TeX (mid 2009) and substantial parts of Con \TeX t have been rewritten using a mixture of Lua and \TeX . In another document, "Con \TeX t MkII--MkIV, the history of Lua \TeX 2006--2009", we have kept track of how both systems evolved so far¹. Here we continue that story which eventually will end with both systems being stable and more or less complete in their basic features.

The title of this document needs some explanation, although the symbols on the cover might give a clue already. In Con \TeX t MkIV, as it is now, we mix several languages:

- good old \TeX : here you will see {} all over the place
- fancy MetaPost: there we use quite some ()
- lean and mean Lua: both {} and () show up a lot there
- unreadable but handy xml: immediately recognizable by the use of <>

As we use all of them mixed, you can consider MkIV to be a hybrid system and just as with hybrid cars, efficiency is part of the concept.



In this graphic we've given Lua a somewhat different place than the other three languages. First of all we have Lua inside \TeX , which is kind of hidden, but at the same time we can use Lua to provide whatever extra features we need, especially when we've reached the state where we can load libraries. In a similar fashion we have utilities (now all written in Lua) that can manage your workflow or aspects of a run (the `mtxrun` script plays a central role in this).

The mentioned history document was (and still is) a rather good testcase for Lua \TeX and MkIV. We explore some new features and load a lot of fonts, some really large. This document will also serve that purpose. This is one of the

¹ Parts of this have been published in usergroup magazines like the Maps, TugBoat, and conference proceedings of Euro \TeX and tug.

reasons why we have turned on grid snapping (and occasionally some tracing).

Keeping track of the history of LuaTeX and MkIV in a document serves several purposes. Of course it shows what has been done. It also serves as a reminder of why it was done that way. As mentioned it serves as test, both in functionality and performance, and as such it's always one of the first documents we run after a change in the code. Most of all this document serves as an extension to my limited memory. When I look at my source code I often can remember when and why it was done that way at that time. However, writing it down more explicitly helps me to remember more and might help users to get some insight in the developments and decisions made.²

Of course, although I wrote most of the text, this document is as much a reflection of what Taco Hoekwater and Hartmut Henkel come up with, but all errors you find here are definitely mine.

Hans Hagen, Hasselt NL,
September 2009 and beyond

<http://www.luatex.org>

² I read a lot and regret that I forget most of what I read so fast. I might as well forget what I wrote so have some patience with me as I repeat myself occasionally.

1 The language mix

During the third ConT_EXt conference that ran in parallel to EuroT_EX 2009 in The Hague we had several sessions where MkIV was discussed and a few upcoming features were demonstrated. The next sections summarize some of that. It's hard to predict the future, especially because new possibilities show up once LuaT_EX is opened up more, so remarks about the future are not definitive.

1.1 T_EX

From now on, if I refer to T_EX in the perspective of LuaT_EX I mean “Good Old T_EX”, the language as well as the functionality. Although LuaT_EX provides a couple of extensions it remains pretty close to compatible to its ancestor, certainly from the perspective of the end user.

As most ConT_EXt users code their documents in the T_EX language, this will remain the focus of MkIV. After all, there is no real reason to abandon it. However, although ConT_EXt already stimulates users to use structure where possible and not to use low level T_EX commands in the document source, we will add a few more structural variants. For instance, we already introduced `\startchapter` and `\startitem` in addition to `\chapter` and `\item`.

We even go further, by using key/value pairs for defining section titles, bookmarks, running headers, references, bookmarks and list entries at the start of a chapter. And, as we carry around much more information in the (for T_EX so typical) auxiliary data files, we provide extensive control over rendering the numbers of these elements when they are recalled (like in tables of contents). So, if you really want to use different texts for all references to a chapter header, it can be done:

```
\startchapter
  [label=emcsquare,
   title={About  $e=mc^2$ },
   bookmark={einstein},
   list={About  $e=mc^2$  (Einstein)},
   reference={ $e=mc^2$ }]
... content ...
\stopchapter
```

Under the hood, the MkIV code base is becoming quite a mix and once we have

a more clear picture of where we're heading, it might become even more of a hybrid. Already for some time most of the font handling is done by Lua, and a bit more logic and management might move to Lua as well. However, as we want to be downward compatible we cannot go as far as we want (yet). This might change as soon as more of the primitives have associated Lua functions. Even then it will be a trade off: calling Lua takes some time and it might not pay off at all.

Some of the more tricky components, like vertical spacing, grid snapping, balancing columns, etc. are already in the process of being Luaified and their hybrid form might turn into complete Lua driven solutions eventually. Again, the compatibility issue forces us to follow a stepwise approach, but at the cost of (quite some) extra development time. But whatever happens, the $\text{T}_{\text{E}}\text{X}$ input language as well as machinery will be there.

1.2 MetaPost

I never regret integrating MetaPost support in Con $\text{T}_{\text{E}}\text{X}$ t and a dream came true when mplib became part of Lua $\text{T}_{\text{E}}\text{X}$. Apart from a few minor changes in the way text integrates into MetaPost graphics the user interface in MkIV is the same as in MkII. Insofar as Lua is involved, this is hidden from the user. We use Lua for managing runs and conversion of the result to pdf. Currently generating MetaPost code by Lua is limited to assisting in the typesetting of chemical structure formulas which is now part of the core.

When defining graphics we use the MetaPost language and not some $\text{T}_{\text{E}}\text{X}$ -like variant of it. Information can be passed to MetaPost using special macros (like `\MPcolor`), but most relevant status information is passed automatically anyway.

You should not be surprised if at some point we can request information from $\text{T}_{\text{E}}\text{X}$ directly, because after all this information is accessible. Think of something `w := texdimen(0) ;` being expanded at the MetaPost end instead of `w := \the\dimen0 ;` being passed to MetaPost from the $\text{T}_{\text{E}}\text{X}$ end.

1.3 Lua

What will the user see of Lua? First of all he or she can use this scripting language to generate content. But when making a format or by looking at the statistics printed at the end of a run, it will be clear that Lua is used all over the place.

So how about Lua as a replacement for the $\text{T}_{\text{E}}\text{X}$ input language? Actually, it is already possible to make such “Con $\text{T}_{\text{E}}\text{X}$ t Lua Documents” using MkIV's built

in functions. Each ConT_EXt command is also available as a Lua function.

```
\startluacode
context.bTABLE {
  framecolor = "blue",
  align= "middle",
  style = "type",
  offset=".5ex",
}
for i=1,10 do
  context.bTR()
  for i=1,20 do
    local r= math.random(99)
    if r < 50 then
      context.bTD {
        background = "color",
        backgroundcolor = "blue"
      }
      context(context.white("%#2i",r))
    else
      context.bTD()
      context("%#2i",r)
    end
    context.eTD()
  end
  context.eTR()
end
context.eTABLE()
\stopluacode
```

Of course it helps if you know ConT_EXt a bit. For instance we can as well say:

```
if r < 50 then
  context.bTD {
    background = "color",
    backgroundcolor = "blue",
    foregroundcolor = "white",
  }
else
  context.bTD()
end
context("%#2i",r)
context.eTD()
```

And, knowing Lua helps as well, since the following is more efficient:

```
\startluacode
  local colored = {
    background = "color",
    backgroundcolor = "blue",
    foregroundcolor = "white",
  }
  local basespec = {
    framecolor = "blue",
    align= "middle",
    style = "type",
    offset=".5ex",
  }
  local bTR, eTR = context.bTR, context.eTR
  local bTD, eTD = context.bTD, context.eTD
  context.bTABLE(basespec)
  for i=1,10 do
    bTR()
    for i=1,20 do
      local r= math.random(99)
      bTD((r < 50 and colored) or nil)
      context("%#2i",r)
      eTD()
    end
    eTR()
  end
  context.eTABLE()
\stopluacode
```

Since in practice the speedup is negligible and the memory footprint is about the same, such optimization seldom make sense.

At some point this interface will be extended, for instance when we can use \TeX 's main (scanning, parsing and processing) loop as a so-called coroutine and when we have opened up more of \TeX 's internals. Of course, instead of putting this in your \TeX source, you can as well keep the code at the Lua end.

The script that manages a Con \TeX t run (also called context) will process files with that consist of such commands directly if they have a `cld` suffix or when you provide the flag `--forcecld`.³

³ Similar methods exist for processing xml files.

34	6	13	46	76	81	71	56	62	19	9	79	55	77	69	57	28	68	60	4
98	38	9	84	71	80	43	33	39	22	51	49	70	85	87	30	42	98	82	6
35	26	42	10	57	97	76	51	17	35	47	32	87	11	79	80	27	74	21	30
56	63	99	13	73	34	58	21	9	45	15	74	3	97	42	69	3	5	89	45
61	62	66	20	33	6	1	70	63	32	76	80	51	49	88	33	78	65	11	83
8	49	4	84	20	88	68	95	4	80	78	48	62	71	80	2	57	87	9	72
53	66	97	44	75	47	41	61	52	6	84	23	49	8	84	49	74	67	38	65
53	76	21	57	50	98	7	54	57	82	41	95	93	6	27	18	60	89	85	42
87	1	73	84	21	81	97	73	78	21	21	96	19	9	93	62	83	95	14	91
11	86	41	38	14	91	60	41	14	74	51	10	8	10	24	55	89	99	88	50

Figure 1.1 The result of the shown Lua code.

```
context yourfile.cld
```

But will this replace \TeX as an input language? This is quite unlikely because coding documents in \TeX is so convenient and there is not much to gain here. Of course in a pure Lua based workflow (for instance publishing information from databases) it would be nice to code in Lua, but even then it's mostly syntactic sugar, as \TeX has to do the job anyway. However, eventually we will have a quite mature Lua counterpart.

1.4 xml

This is not so much a programming language but more a method of tagging your document content (or data). As structure is rather dominant in xml, it is quite handy for situations where we need different output formats and multiple tools need to process the same data. It's also a standard, although this does not mean that all documents you see are properly structured. This in turn means that we need some manipulative power in $\text{Con}\text{\TeX}t$, and that happens to be easier to do in MkIV than in MkII .

In $\text{Con}\text{\TeX}t$ we have been supporting xml for a long time, and in MkIV we made the switch from stream based to tree based processing. The current implementation is mostly driven by what has been possible so far but as $\text{Lua}\text{\TeX}$ becomes more mature, bits and pieces will be reimplemented (or at least cleaned up and brought up to date with developments in $\text{Lua}\text{\TeX}$).

One could argue that it makes more sense to use xslt for converting xml into something \TeX , but in most of the cases that I have to deal with much effort

goes into mapping structure onto a given layout specification. Adding a bit of xml to $\text{T}_{\text{E}}\text{X}$ mapping to that directly is quite convenient. The total amount of code is probably smaller and it saves a processing step.

We're mostly dealing with education-related documents and these tend to have a more complex structure than the final typeset result shows. Also, readability of code is not served with such a split as most mappings look messy anyway (or evolve that way) due to the way the content is organized or elements get abused.

There is a dedicated manual for dealing with xml in MkIV, so we only show a simple example here. The documents to be processed are loaded in memory and serialized using setups that are associated to elements. We keep track of documents and nodes in a way that permits multipass data handling (rather usual in $\text{T}_{\text{E}}\text{X}$). Say that we have a document that contains questions. The following definitions will flush the (root element) questions:

```
\startxmlsetups xml:mysetups
  \xmlsetsetup{#1}{questions}{xml:questions}
\stopxmlsetups

\xmlregistersetup{xml:mysetups}

\startxmlsetups xml:questions
  \xmlflush{#1}
\stopxmlsetups

\xmlprocessfile{main}{somefile.xml}{}
```

Here the #1 represents the current xml element. Of course we need more associations in order to get something meaningful. If we just serialize then we have mappings like:

```
\xmlsetsetup{#1}{question|answer}{xml:*}
```

So, questions and answers are mapped onto their own setup which flushes them, probably with some numbering done at the spot.

In this mechanism Lua is sort of invisible but quite busy as it is responsible for loading, filtering, accessing and serializing the tree. In this case $\text{T}_{\text{E}}\text{X}$ and Lua hand over control in rapid succession.

You can hook in your own functions, like:


```
\xmlfilter{#1}{(wording|feedback|choice)/function(cleanup)}
```

In this case the function `cleanup` is applied to elements with names that match one of the three given.⁴

Of course, once you start mixing in Lua in this way, you need to know how we deal with xml at the Lua end. The following function show how we calculate scores:

```
\startluacode
function xml.functions.totalscore(root)
  local n = 0
  for e in xml.collected(root, "/outcome") do
    if xml.filter(e, "action[text()='add']") then
      local m = xml.filter(e, "xml:///score/text()")
      n = n + (tonumber(m or 0) or 0)
    end
  end
  tex.write(n)
end
\stopluacode
```

You can either use such a function in a filter or just use it as a \TeX macro:

```
\startxmlsetups xml:question
  \blank
  \xmlfirst{#1}{wording}
  \startitemize
    \xmlfilter{#1}{/answer/choice/command(xml:answer:choice)}
  \stopitemize
  \endgraf
  score: \xmlfunction{#1}{totalscore}
  \blank
\stopxmlsetups

\startxmlsetups xml:answer:choice
  \startitem
    \xmlflush{#1}
  \stopitem
\stopxmlsetups
```

⁴ This example is inspired by one of our projects where the cleanup involves sanitizing (highly invalid) html data that is embedded as a CDATA stream, a trick to prevent the xml file to be invalid.

The filter variant is like this:

```
\xmlfilter{#1}{./function('totalscore')}
```

So you can take your choice and make your source look more xml-ish, Lua-like or \TeX -wise. A careful reader might have noticed the peculiar `xml://` in the function code. When used inside MkIV, the serializer defaults to \TeX so results are piped back into \TeX . This prefix forced the regular serializer which keeps the result at the Lua end.

Currently some of the xml related modules, like MathML and handling of tables, are really a mix of \TeX code and Lua calls, but it makes sense to move them completely to Lua. One reason is that their input (formulas and table content) is restricted to non- \TeX anyway. On the other hand, in order to be able to share the implementation with \TeX input, it also makes sense to stick to some hybrid approach. In any case, more of the calculations and logic will move to Lua, while \TeX will deal with the content.

A somewhat strange animal here is `xsl-fo`. We do support it, but the MkII implementation was always somewhat limited and the code was quite complex. So, this needs a proper rewrite in MkIV, which will happen indeed. It's mostly a nice exercise of hybrid technology but until now I never really needed it. Other bits and pieces of the current xml goodies might also get an upgrade.

There is already a bunch of functions and macros to filter and manipulate xml content and currently the code involved is being cleaned up. What direction we go also depends on users' demands. So, with respect to xml you can expect more support, a better integration and an upgrade of some supported xml related standards.

1.5 Tools

Some of the tools that ship with Con \TeX t are also examples of hybrid usage.

Take this:

```
mtxrun --script server --auto
```

On my machine this reports:

```
MTXrun | running at port: 31415  
MTXrun | document root: c:/data/develop/context/lu  
MTXrun | main index file: unknown
```

```
MTXrun | scripts subpath: c:/data/develop/context/lua
MTXrun | context services: http://localhost:31415/mtx-server-ctx-startup.lua
```

The `mtxrun` script is a Lua script that acts as a controller for other scripts, in this case `mtx-server.lua` that is part of the regular distribution. As we use Lua \TeX as a Lua interpreter and since Lua \TeX has a socket library built in, it can act as a web server, limited but quite right for our purpose.⁵

The web page that pops up when you enter the given address lets you currently choose between the Con \TeX t help system and a font testing tool. In figure 1.2 you seen an example of what the font testing tool does.



Figure 1.2 An example of using the font tester.

Here we have Lua \TeX running a simple web server but it's not aware of having \TeX on board. When you click on one of the buttons at the bottom of the screen, the server will load and execute a script related to the request and in this case that script will create a \TeX file and call Lua \TeX with Con \TeX t to process that file. The result is piped back to the browser.

You can use this tool to investigate fonts (their bad and good habits) as well as to test the currently available OpenType functionality in MkIV (bugs as well as goodies).

⁵ This application is not intentional but just a side effect.

So again we have a hybrid usage although in this case the user is not confronted with Lua and/or TeX at all. The same is true for the other goodie, shown in figure 1.3. Actually, such a goodie has always been part of the ConTeXt distribution but it has been rewritten in Lua.

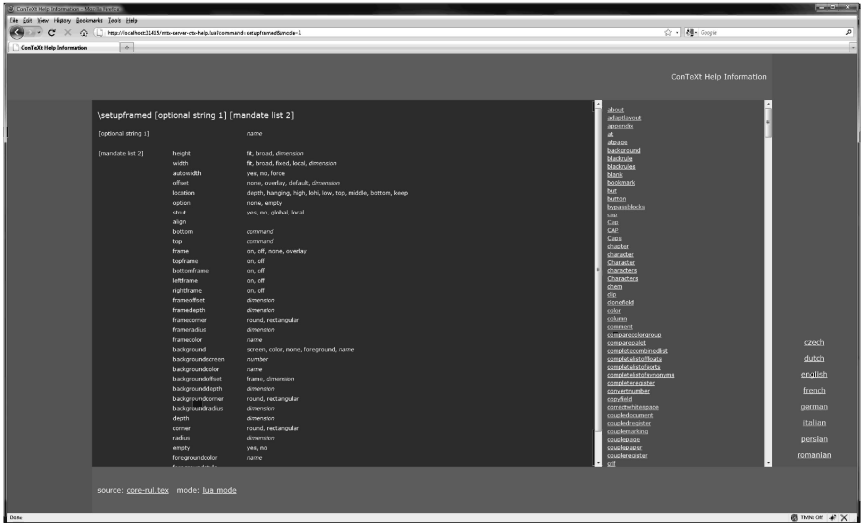


Figure 1.3 An example of a help screen for a command.

The ConTeXt user interface is defined in an xml file, and this file is used for several purposes: initializing the user interfaces at format generation time, type-setting the formal command references (for all relevant interface languages), for the wiki, and for the mentioned help goodie.

Using the mix of languages permits us to provide convenient processing of documents that otherwise would demand more from the user than it does now. For instance, imagine that we want to process a series of documents in the so-called Epub format. Such a document is a zipped file that has a description and resources. As the content of this archive is prescribed it's quite easy to process it:

```
context --ctx=x-epub.ctx yourfile.epub
```

This is equivalent to:

```
texlua mtxrun.lua --script context --ctx=x-epub.ctx yourfile.epub
```

So, here we have Lua \TeX running a script that itself (locates and) runs a script context. That script loads a Con \TeX t job description file (with suffix `ctx`). This file tells what styles to load and might have additional directives but none of that has to bother the end user. In the automatically loaded style we take care of reading the xml files from the zipped file and eventually map the embedded html like files onto style elements and produce a pdf file. So, we have Lua managing a run and MkIV managing with help of Lua reading from zip files and converting xml into something that \TeX is happy with. As there is no standard with respect to the content itself, i.e. the rendering is driven by whatever kind of structure is used and whatever the css file is able to map it onto, in practice we need an additional style for this class of documents. But anyway it's a good example of integration.

1.6 The future

Apart from these language related issues, what more is on the agenda? To mention a few integration related thoughts:

- At some point I want to explore the possibility to limit processing to just one run, for instance by doing trial runs without outputting anything but still collecting multipass information. This might save some runtime in demanding workflows especially when we keep extensive font loading and image handling in mind.
- Related to this is the ability to run MkIV as a service but that demands that we can reset the state of Lua \TeX and actually it might not be worth the trouble at all given faster processors and disks. Also, it might not save much runtime on larger jobs.
- More interesting can be to continue experimenting with isolating parts of Con \TeX t in such a way that one can construct a specialized subset of functionality. Of course the main body of code will always be loaded as one needs basic typesetting anyway.

Of course we keep improving existing mechanisms and improve solutions using a mix of \TeX and Lua, using each language (and system) for what it can do best.

2 Font Goodies

2.1 Introduction

The Oriental \TeX project is one of the first and more ambitious users of Lua \TeX . A major undertaking in this project is the making of a rather full features and complex font for typesetting Arabic. As the following text will show some Arabic, you might get the impression that I'm an expert but be warned that I'm far from that. But as Idris compensates this quite well the team has a lot of fun in figuring out how to achieve our goals using OpenType technology in combination with Lua \TeX and MkIV. A nice side effect of this is that we end up with some neat tricks in the Con \TeX t core.

Before we come to some of these goodies, an example of Arabic is given that relates quite well to the project. It was first used at the euro \TeX 2009 meeting. Take the following 6 shapes:

خ ي ت ا و ل
l w ā t ī kh

With these we can make the name Lua \TeX and as we use a nice script we can forget about the lowered E. Putting these characters in sequence is not enough as Arabic typesetting has to mimick the subtle aspects of scribes.

In Latin scripts we have mostly one-to-one and many-to-one substitutions. These can happen in sequence which in practice boils down to multiple passed over the stream of characters. In this process sometimes surrounding characters (or shapes) play a role, for instance ligatures are not always wanted and their coming into existence might depend on neighbouring characters. In some cases glyphs have to be (re)positioned relative to each other. While in Latin scripts the number of substitutions and positioning is not that large but in advanced Arabic fonts it can be pretty extensive.

With OpenType we have some machinery available, so we try to put as much logic in the font as possible. However, in addition we have some dedicated optimizing routines. The whole process is split into a couple of stages.

The so called First-Order Analysis puts a given character into isolated, initial, middle, or final state. Next, the Second-Order Analysis looks at the characters and relates this state to what characters precede or succeed it. Based on that state we do character substitutions. There can be multiple analysis and replacements in sequence. We can do some simple aesthetic stretching and

additional related replacements. We need to attach identity marks and vowels in proper but nice looking places. In most cases we're then done. Contrary to other fonts we don't use many ligatures but compose characters.

The previous steps already give reasonable results and implementing it also nicely went along with the development of LuaTeX and ConTeXt MkIV. Currently we're working on extending and perfecting the font to support what we call Third-Order Contextual Analysis. This boils down to an interplay between the paragraph builder and additional font features. In order to get pleasing spacing we apply further substitutions, this time with wider or narrower shapes. When this is done we need to reattach identity marks and vowels. Optionally we can apply hz like stretching as a finishing touch but so far we didn't follow that route yet.

So, let's see how we can typeset the word LuaTeX in Arabic using some of these techniques.

no order (kh ī t ā w [u] l)

لُواتِيخ

first order

لُواتِيخ

second order

لُوائِيخ

second order (Jiim-stacking)

لُوائِيخ

minimal stretching

لُوائِيخ

maximal stretching (level 3)

لُوائِيخ

chopped letter khaa (for e.g. underlining)

لُوائِيخ

As said, this font is quite complex in the sense that it has many features and associated lookups. In addition to the usual features we have stylistic and justification variants. As these are not standardized (after all, each font can have its own look and feel and associated treatments) we store some information in the goodies files that ships with this font.

feature	meaning
js01	Raawide
js02	Yaawide
js03	Kaafwide
js04	Nuunwide
js05	Kaafwide Nuunwide Siinwide Baawide
js06	final Haa wide
js07	thin Miim
js08	short Miim
js09	wide Siin
js10	thuluth-style initial Haa, final Miim, MRw_mf
js11	level-1 stretching
js12	level-2 stretching
js13	level-3 stretching
js14	final Alif
js15	hooked final Alif
js16	aesthetic medial Faa/Qaaf
js17	fancy isol Haa after Daal, Raa, and Waaw
js18	Laamwide, alternate substitution
js19	level-4 stretching, only siin and Hhaa for basmalah
js20	level-5 stretching, only siin and Hhaa for basmalah
js21	Haa.final_alt2
ss01	Allah, Muhammad
ss02	ss01 + Allah_final
ss03	level-1 stack over Jiim, initial entry only
ss04	level-1 stack over Jiim, initial/medial entry
ss05	multi-level Jiim stacking, initial/medial entry
ss06	aesthetic Faa/Qaaf for FJ_mm, FJ_mf connection
ss07	initial-entry stacking over Haa
ss08	initial/medial stacking over Haa, minus HM_mf strings
ss09	initial/medial Haa stacking plus HM_mf strings
ss10	basic dipped Miim, initial-entry B_S-stack over Miim
ss11	full dipped Miim, initial-entry B_S-stack over Miim
ss12	XBM_im initial-medial entry B_S-stack over Miim
ss13	full initial-medial entry B_S-stacked Miim
ss14	initial entry, stacked Laam on Miim
ss15	full stacked Laam-on-Miim
ss16	initial entry, stacked Ayn-on-Miim

ss17 full stacked Ayn-on-Miim
 ss18 LMJ_im already contained in ss03--05, may remove
 ss19 LM_im
 ss20 KLM_m, sloped Miim
 ss21 KLM_i_mm/LM_mm, sloped Miim
 ss22 filled sloped Miim
 ss23 LM_mm, non-sloped Miim
 ss24 BR_i_mf, BN_i_mf
 ss25 basic LH_im might merge with ss24
 ss26 full Yaa.final special strings: BY_if, BY_mf, LY_mf
 ss27 basic thin Miim.final
 ss28 full thin Miim.final to be moved to jsnn
 ss29 basic short Miim.final
 ss30 full short Miim.final to be moved to jsnn
 ss31 basic Raa.final strings: JR and SR
 ss32 basic Raa.final strings: JR, SR, and BR
 ss33 TtR to be moved to jsnn
 ss34 AyR style also available in jsnn
 ss35 full Kaaf contexts
 ss36 full Laam contexts
 ss37 Miim-Miim contexts
 ss38 basic dipped Haa, B_SH_mm
 ss39 full dipped Haa, B_S_LH_i_mm_Mf
 ss40 aesthetic dipped medial Haa
 ss41 high and low Baa strings
 ss42 diagonal entry
 ss43 initial alternates
 ss44 hooked final alif
 ss45 BMA_f
 ss46 BM_mm_alt, for JBM combinations
 ss47 Shaddah-<kasrah> combo
 ss48 Auto-sukuun
 ss49 No vowels
 ss50 Shaddah/MaaddahHamzah only
 ss51 No Skuun
 ss52 No Waslah
 ss53 No Waslah
 ss54 chopped finals
 ss55 idgham-tanwin

It is highly unlikely that a user will remember all these features, which is why there will be a bunch of predefined combinations. These are internalized as follows:

featureset	definitions
default	analyze=true anum=true calt=true ccmp=true curs=true fina=true init=true js16=true kern=true language=dflt mark=true medi=true mkmk=true mode=node number=32 rlig=true salt=true script=arab ss01=true ss03=true ss07=true ss10=true ss12=true ss15=true ss16=true ss19=true ss24=true ss25=true ss26=true ss27=true ss31=true ss34=true ss35=true ss36=true ss37=true ss38=true ss41=true ss42=true ss43=true
maximal_stretching	analyze=true anum=true calt=true ccmp=true curs=true fina=true init=true js05=true js09=true js13=true js16=true kern=true language=dflt mark=true medi=true mkmk=true mode=node number=35 rlig=true salt=true script=arab ss01=true ss03=true ss07=true ss10=true ss12=true ss15=true ss16=true ss19=true ss24=true ss25=true ss26=true ss27=true ss31=true ss34=true ss35=true ss36=true ss37=true ss38=true ss41=true ss42=true ss43=true
medium_stretching	analyze=true anum=true calt=true ccmp=true curs=true fina=true init=true js05=true js12=true js16=true kern=true language=dflt mark=true medi=true mkmk=true mode=node number=36 rlig=true salt=true script=arab ss01=true ss03=true ss07=true ss10=true ss12=true ss15=true ss16=true ss19=true ss24=true ss25=true ss26=true ss27=true ss31=true ss34=true ss35=true ss36=true ss37=true ss38=true ss41=true ss42=true ss43=true
minimal_stretching	analyze=true anum=true calt=true ccmp=true curs=true fina=true init=true js03=true js11=true js16=true kern=true language=dflt mark=true medi=true mkmk=true mode=node number=31 rlig=true salt=true script=arab ss01=true ss03=true ss07=true ss10=true ss12=true ss15=true ss16=true ss19=true ss24=true ss25=true ss26=true ss27=true ss31=true ss34=true ss35=true ss36=true ss37=true ss38=true ss41=true ss42=true ss43=true

```

shrink      analyze=true anum=true calt=true ccmp=true
            curs=true  fina=true  flts=true  init=true
            js16=true  js17=true kern=true  language=dflt
            mark=true  medi=true  mkmk=true  mode=node
            number=34  rlig=true  salt=true  script=arab
            ss01=true  ss03=true  ss05=true  ss06=true
            ss07=true  ss09=true  ss10=true  ss11=true
            ss12=true  ss15=true  ss16=true  ss19=true
            ss24=true  ss25=true  ss26=true  ss27=true
            ss31=true  ss34=true  ss35=true  ss36=true
            ss37=true  ss38=true  ss41=true  ss42=true
            ss43=true

wide_all    analyze=true anum=true calt=true ccmp=true
            curs=true  fina=true  init=true  js05=true
            js09=true  js11=true  js12=true  js13=true
            js16=true  kern=true  language=dflt mark=true
            medi=true  mkmk=true  mode=node  number=33
            rlig=true  salt=true  script=arab ss01=true
            ss03=true  ss07=true  ss10=true  ss12=true
            ss15=true  ss16=true  ss19=true  ss24=true
            ss25=true  ss26=true  ss27=true  ss31=true
            ss34=true  ss35=true  ss36=true  ss37=true
            ss38=true  ss41=true  ss42=true  ss43=true

```

2.2 Color

One of the objectives of the oriental $\text{T}_{\text{E}}\text{X}$ project is to bring color to typeset Arabic. When Idris started making samples with much manual intervention it was about time to figure out if it could be supported by a bit of Lua code.

As the colorization concerns classes of glyphs (like vowels) this is something that can best be done after all esthetics have been sorted out. Because things like coloring are not part of font technology and because we don't want to misuse the OpenType feature mechanisms for that, the solution lays in an extra file that describes these goodies.

لوائخ ألف ليلة وليلة

لُوَانِيحُ أَلْفُ لَيْلَةٍ وَلَيْلَةٍ

لُوَانِيحُ أَلْفُ لَيْلَةٍ وَلَيْلَةٍ

The second and third of these three lines have colored vowels and identity marks. So how did we get the colors? There are actually two mechanisms involved in this:

- we need to associate colorschemes with classed of glyphs
- we need to be able to turn on and off coloring

The first is done by loading goodies and selecting a colorscheme:

```
\definefontfeature  
  [husayni-colored]  
  [goodies=husayni,  
   colorscheme=default,  
   featureset=default]
```

Turning on and off coloring is done with two commands (we might provide a proper environment for this) as shown in:

```
\start  
  \definedfont[husayni*husayni-colored at 72pt]  
  \righttoleft  
  \resetfontcolorscheme  لُوَانِيحُ أَلْفُ لَيْلَةٍ وَلَيْلَةٍ \par  
  \setfontcolorscheme  [1]لُوَانِيحُ أَلْفُ لَيْلَةٍ وَلَيْلَةٍ \crlf  
  \setfontcolorscheme  [2]لُوَانِيحُ أَلْفُ لَيْلَةٍ وَلَيْلَةٍ \crlf  
\stop
```

If you look closely at the feature definition you'll notice that we also choose a default featureset. For most (latin) fonts the regular feature definitions are convenient, but for fonts that are used for Arabic there are preferred combinations of features as there can be many.

Currently the font we use here has the following colorschemes:

```
colorscheme numbers
default      1 2 3 4 5
```

2.3 The goodies file

In principle a goodies files can contain any data that makes sense but in order to be useable some entries have a prescribed structure. A goodies file looks as follows:

```
return {
  name = "husayni",
  version = "1.00",
  comment = "Goodies that complement the Husayni font by Idris Samawi Hamid.",
  author = "Idris Samawi Hamid and Hans Hagen",
  featuresets = {
    default = {
      key = value, <table>, ...
    },
    ...
  },
  stylistics = {
    key = value, ...
  },
  colorschemes = {
    default = {
      [1] = {
        "glyph_a.one", "glyph_b.one", ...
      },
      ...
    }
  }
}
```

We already saw the list of special features and these are defined in the `stylistics` stable. In this document, that list was typeset using the following (hybrid) code:

```
\startluacode
  local goodies = fonts.goodies.get("husayni")
  local stylistics = goodies and goodies.stylistics
  if stylistics then
    local col, row, type = context.NC, context.NR, context.type
    context.starttabulate { "|l|p|l|" }
```

```

    col() context("feature") col() context("meaning") col() row()
    for feature, meaning in table.sortedpairs(stylistics) do
      col() type(feature) col() type(meaning) col() row()
    end
    context.stoptabulate()
  end
\stoptluacode

```

The table with colorscheme that we showed is generated with:

```

colorscheme numbers
default      1 2 3 4 5

```

In a similar fashion we typeset the featuresets:

```

\startluacode
  local goodies = fonts.goodies.get("husayni")
  local featuresets = goodies and goodies.featuresets
  if featuresets then
    local col, row, type = context.NC, context.NR, context.type
    context.starttabulate { "|l|pl|" }
    col() context("featureset") col() context("definitions") col() row()
    for featureset, definitions in table.sortedpairs(featuresets) do
      col() type(featureset) col()
      for k, v in table.sortedpairs(definitions) do
        type(string.format("%s=%s",k,tostring(v)))
        context.quad()
      end
      col() row()
    end
    context.stoptabulate()
  end
\stoptluacode

```

The unprocessed featuresets table can contain one or more named sets and each set can be a mixture of tables and key value pairs. Say that we have:

```

default = {
  kern = "yes", { ss01 = "yes" }, { ss02 = "yes" }, "mark"
}

```

Given the previous definition, the order of processing is as follows.

1. { ss01 = "yes" }
2. { ss02 = "yes" }
3. mark (set to "yes")
4. kern = "yes"

So, first we process the indexed part if the list, and next the hash. Already set values are not set again. The advantage of using a Lua table is that you can simplify definitions. Before we return the table we can define local variables, like:

```
local one = { ss01 = "yes" }
local two = { ss02 = "yes" }
local pos = { kern = "yes", mark = "yes" }
```

and use them in:

```
default = {
  one, two, pos
}
```

That way we we can conveniently define all kind of interesting combinations without the need for many repetitive entries.

The `colorsets` table has named subtables that are (currently) indexed by number. Each number is associated with a color (at the \TeX end) and is coupled to a list of glyphs. As you can see here, we use the name of the glyph. We prefer this over an index (that can change during development of the font). We cannot use Unicode points as many such glyphs are just variants and have no unique code.

2.4 Optimizing Arabic

The ultimate goal of the Oriental \TeX project is to improve the look and feel of a paragraph. Because \TeX does a pretty good job on breaking the paragraph into lines, and because complicating the paragraph builder is not a good idea, we finally settled on improving the lines that result from the par builder. This approach is rather close to what scribes do and the advanced Husayni font provides features that support this.

In principle the current optimizer can replace character expansion but that would slow down considerably. Also, for that we first have to clean up the experimental Lua based par builder.

After several iterations the following approach was chosen.

- We typeset the paragraph with an optimal feature set. In our case this is `husayni-default`.
- Next we define two sets of additional features: one that we can apply to shrink words, and one that does the opposite.
- When the line has a badness we don't like, we either stepwise shrink words or stretch them, depending on how bad things are.

The set that takes care of shrinking is defined as:

```
\definefontfeature
  [shrink]
  [husayni-default]
  [flts=yes, js17=yes, ss05=yes, ss11=yes, ss06=yes, ss09=yes]
```

Stretch has a few more variants:

```
\definefontfeature
  [minimal_stretching]
  [husayni-default]
  [js11=yes, js03=yes]
\definefontfeature
  [medium_stretching]
  [husayni-default]
  [js12=yes, js05=yes]
\definefontfeature
  [maximal_stretching]
  [husayni-default]
  [js13=yes, js05=yes, js09=yes]
\definefontfeature
  [wide_all]
  [husayni-default]
  [js11=yes, js12=yes, js13=yes, js05=yes, js09=yes]
```

Next we define a font solution:

```
\definefontsolution
  [FancyHusayni]
  [goodies=husayni,
   less=shrink,
   more={minimal_stretching,medium_stretching,maximal_stretching,wide_all}]
```

Because these featuresets relate quite close to the font design we don't use this

way if defining but put the definitions in the goodies file:

```
.....
featuresets = { -- here we don't have references to featuresets
  default = {
    default,
  },
  minimal_stretching = {
    default, js11 = yes, js03 = yes,
  },
  medium_stretching = {
    default, js12=yes, js05=yes,
  },
  maximal_stretching= {
    default, js13 = yes, js05 = yes, js09 = yes,
  },
  wide_all = {
    default, js11 = yes, js12 = yes, js13 = yes, js05 = yes, js09 = yes,
  },
  shrink = {
    default, flts = yes, js17 = yes, ss05 = yes, ss11 = yes, ss06 = yes, ss09 = yes,
  },
},
solutions = { -- here we have references to featuresets, so we use strings!
  experimental = {
    less = { "shrink" },
    more = { "minimal_stretching", "medium_stretching", "maximal_stretching", "wide_all" },
  },
},
.....
```

Now the definition looks much simpler:

```
\definefontsolution
[FancyHusayni]
[goodies=husayni,
 solution=experimental]
```

I want some funny text (complete with translation). Actually I want all examples translated.

In the following example the yellow words are stretched and the green ones are shrunken.⁶

```
\definedfont[husayni*husayni-default at 24pt]
\expanded{\setuplocalinterlinespace[line=\the\dimexpr2\lineheight]} % todo:
```

⁶ Make sure that the paragraph is finished (for instance using `\par` before resetting it.)

```

factor ivm grid
\setfontsolution[FancyHusayni]% command will change
\enabletrackers[builders.paragraphs.solutions.splitters.colors]
\righttoleft \getbuffer[sample] \par
\disabletrackers[builders.paragraphs.solutions.splitters.colors]
\resetfontsolution

```

قد صعدا ذرى الحقائق بأقدام النبوة والولاية ونورنا سيع طبقات أعلام الفتوى بالهداية فنحن ليوث الوغى وغيوث الندى وطعان العدى وفينا السيف والقلم في العاجل ولواء الحمد والحوض في الآجل وأسباطنا حلفاء الدين وخلفاء النبيين ومصايح الأمم ومفاتيح الكرم فالكلم ألبس حلة الاصطفاء لما عهدنا منه الوفاء وروح القدس في جنان الصاقورة ذاق من حدائقنا الباكورة وشيعتنا الفئة الناجية والفرقة الزاكية وصاروا لنا رداء وصونا وعلى الظلة ألبا وعونا وسينفجر لهم بناييع الحيوان بعد لظى النهران لثام آل حم وطه والطواسين من السنين وهذا الكتاب درة من درر الرحمة وقطرة من بحر الحكمة وكتب الحسن بن علي العسكري في سنة أربع وخمسين ومائتين

This mechanism is somewhat experimental as is the (user) interface. It is also rather slow compared to normal processing. There is room for improvement but I will do that when other components are more stable so that simple variants (that we can use here) can be derived.

When criterium 0 used above is changed into for instance 5 processing is faster. When you enable a preroll processing is more time consuming. Examples of settings are:

```

\setupfontolutions[method={preroll,normal},criterium=2]
\setupfontolutions[method={preroll,random},criterium=5]
\setupfontolutions[method=reverse,criterium=8]
\setupfontolutions[method=random,criterium=2]

```

Using a preroll is slower because it first tries all variants and then settles for the best; otherwise we process the first till the last solution till the criterium

is satisfied.

Todo: show normal, reverse and random.

Todo: bind setting to paragraph.

2.5 Protrusion and expansion

There are two entries in the goodies file that relate to advanced parbuilding: protrusions and expansions.

```
protrusions = {  
  vectors = {  
    pure = {  
      [0x002C] = { 0, 1 }, -- comma  
      [0x002E] = { 0, 1 }, -- period  
      .....  
    }  
  }  
}
```

These vectors are similar to the ones defined globally but the vectors defined in a goodie file are taken instead when present.

3 Grouping

3.1 Variants

After using \TeX for a while you get accustomed to one of its interesting concepts: grouping. Programming languages like Pascal and Modula have keywords `begin` and `end`. So, one can say:

```
if test then begin
    print_bold("test 1")
    print_bold("test 2")
end
```

Other languages provide a syntax like:

```
if test {
    print_bold("test 1")
    print_bold("test 2")
}
```

So, in those languages the `begin` and `end` and/or the curly braces define a ‘group’ of statements. In \TeX on the other hand we have:

```
test \begingroup \bf test \endgroup test
```

Here the second `test` comes out in a bold font and the switch to bold (basically a different font is selected) is reverted after the group is closed. So, in \TeX grouping deals with scope and not with grouping things together.

In other languages it depends on the language of locally defined variables are visible afterwards but in \TeX they’re really local unless a `\global` prefix (or one of the shortcuts) is used.

In languages like Lua we have constructs like:

```
for i=1,100 do
    local j = i + 20
    ...
end
```

Here `j` is visible after the loop ends unless prefixed by `local`. Yet another example is MetaPost:

```
begingroup ;
```

```

    save n ; numeric n ; n := 10 ;
    ...
endgroup ;

```

Here all variables are global unless they are explicitly saved inside a group. This makes perfect sense as the resulting graphic also has a global (accumulated) property. In practice one will rarely needs grouping, contrary to \TeX where one really wants to keep changes local, if only because document content is so unpredictable that one never knows when some change in state happens.

In principle it is possible to carry over information across a group boundary. Consider this somewhat unrealistic example:

```

\begingroup
  \leftskip 10pt
  \begingroup
    ....
    \advance\leftskip 10pt
    ....
  \endgroup
\endgroup

```

How do we carry the advanced leftskip over the group boundary without using a global assignment which could have more drastic side effects? Here is the trick:

```

\begingroup
  \leftskip 10pt
  \begingroup
    ....
    \advance\leftskip 10pt
    ....
    \expandafter
  \endgroup
  \expandafter \leftskip \the\leftskip
\endgroup

```

This is typical the kind of code that gives new users the creeps but normally they never have to do that kind of coding. Also, that kind of tricks assumes that one knows how many groups are involved.

3.2 Implication

What does this all have to do with Lua \TeX and MkIV? The user interface of Con \TeX t provide lots of commands like:

```
\setupthis[style=bold]
\setupthat[color=green]
```

Most of them obey grouping. However, consider a situation where we use Lua code to deal with some aspect of typesetting, for instance numbering lines or adding ornamental elements to the text. In Con \TeX t we flag such actions with attributes and often the real action takes place a bit later, for instance when a paragraph or page becomes available.

A comparable pure \TeX example is the following:

```
{test test \bf test \leftskip10pt test}
```

Here the switch to bold happens as expected but no leftskip of 10pt is applied. This is because the set value is already forgotten when the paragraph is actually typeset. So in fact we'd need:

```
{test test \bf test \leftskip10pt test \par}
```

Now, say that we have:

```
{test test test \setupflag[option=1] \flagnexttext test}
```

We flag some text (using an attribute) and expect it to get a treatment where option 1 is used. However, the real action might take place when \TeX deals with the paragraph or page and by that time the specific option is already forgotten or it might have gotten another value. So, the rather natural \TeX grouping does not work out that well in a hybrid situation.

As the user interface assumes a consistent behaviour we cannot simply make these settings global even if this makes much sense in practice. One solution is to carry the information with the flagged text i.e. associate it somehow in the attribute's value. Of course, as we never know in advance when this information is used, this might result in quite some states being stored persistently.

A side effect of this 'problem' is that new commands might get suboptimal user interfaces (especially inheritance or cloning of constructs) that are somewhat driven by these 'limitations'. Of course we may wonder if the end user will notice this.

To summarize this far, we have three sorts of grouping to deal with:

- \TeX 's normal grouping model limits its scope to the local situation and normally has only direct and local consequences. We cannot carry information over groups.
- Some of \TeX 's properties are applied later, for instance when a paragraph or page is typeset and in order to make 'local' changes effective, the user needs to add explicit paragraph ending commands (like `\par` or `\page`).
- Features dealt with asynchronously by Lua are at that time unaware of grouping and variables set that were active at the time the feature was triggered so there we need to make sure that our settings travel with the feature. There is not much that a user can do about it as this kind of management has to be done by the feature itself.

It is the third case that we will give an example of in the next section. We leave it up to the user if it gets noticed on the user interface.

3.3 An example

A group of commands that has been reimplemented using a hybrid solution is underlining or more generic: bars. Just take a look at the following examples and try to get an idea on how to deal with grouping. Keep in mind that:

- Colors are attributes and are resolved in the backend, so way after the paragraph has been typesetting.
- Overstrike is also handled by an attribute and gets applied in the backend as well, before colors are applied.
- Nested overstrikes might have different settings.
- An overstrike rule either inherits from the text or has its own color setting.

First an example where we inherit color from the text:

```
\definecolor[myblue][b=.75]
\definebar[myoverstrike][overstrike][color=]
```

```
Test \myoverstrike{%
  Test \myoverstrike{\myblue
    Test \myoverstrike{Test}
    Test}
  Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Because color is also implemented using attributes and processed later on we can access that information when we deal with the bar.

The following example has its own color setting:

```
\definecolor[myblue][b=.75]
\definecolor[myred][r=.75]
\definebar[myoverstrike][overstrike][color=myred]
```

```
Test \myoverstrike{%
  Test \myoverstrike{\myblue
    Test \myoverstrike{Test}
    Test}
  Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

See how can we color the levels differently:

```
\definecolor[myblue][b=.75]
\definecolor[myred][r=.75]
\definecolor[mygreen][g=.75]

\definebar[myoverstrike:1][overstrike][color=myblue]
\definebar[myoverstrike:2][overstrike][color=myred]
\definebar[myoverstrike:3][overstrike][color=mygreen]
```

```
Test \myoverstrike{%
  Test \myoverstrike{%
    Test \myoverstrike{Test}
    Test}
  Test}
Test
```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Watch this:

```
\definecolor[myblue][b=.75]
\definecolor[myred][r=.75]
\definecolor[mygreen][g=.75]
```



```

\definebar[myoverstrike][overstrike][max=1,dy=0,offset=.5]
\definebar[myoverstrike:1][myoverstrike][color=myblue]
\definebar[myoverstrike:2][myoverstrike][color=myred]
\definebar[myoverstrike:3][myoverstrike][color=mygreen]

```

```

Test \myoverstrike{%
  Test \myoverstrike{%
    Test \myoverstrike{Test}
    Test}
  Test}
Test

```

Test ~~Test~~ Test ~~Test~~ Test Test Test

It this the perfect user interface? Probably not, but at least it keeps the implementation quite simple.

The behaviour of the MkIV implementation is roughly the same as in MkII, although now we specify the dimensions and placement in terms of the ratio of the x-height of the current font.

```

Test \overstrike{Test \overstrike{Test \overstrike{Test} Test} Test} Test
\blank
Test \underbar {Test \underbar {Test \underbar {Test} Test} Test} Test
\blank
Test \overbar {Test \overbar {Test \overbar {Test} Test} Test} Test
\blank
Test \underbar {Test \overbar {Test \overstrike{Test} Test} Test} Test
\blank

```

Test ~~Test~~ ~~Test~~ ~~Test~~ ~~Test~~ Test

Test Test Test Test Test Test

Test Test Test Test Test

Test Test ~~Test~~ Test Test Test

As an extra this mechanism can also provide simple backgrounds. The normal background mechanism uses MetaPost and the advantage is that we can use arbitrary shapes but it also carries some limitations. When the development of LuaTeX is a bit further along the road I will add the possibility to use MetaPost

shapes in this mechanism.

Before we come to backgrounds, first take a look at these examples:

```
\startbar[underbar] \input zapf \stopbar \blank  
\startbar[underbars] \input zapf \stopbar \blank
```

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

First notice that it is no problem to span multiple lines and that hyphenation is not influenced at all. Second you can see that continuous rules are also possible. From such a continuous rule to a background is a small step:

```
\definebar  
  [backbar]  
  [offset=1.5,rulethickness=2.8,color=blue,  
   continue=yes,order=background]
```

```
\definebar  
  [forebar]  
  [offset=1.5,rulethickness=2.8,color=blue,  
   continue=yes,order=foreground]
```



The following example code looks messy but this has to do with the fact that we want properly spaced sample injection.

```
from here  
  \startcolor[white]%
```

```

\startbar[backbar]%
  \input zapf
  \removeunwantedspaces
\stopbar
\stopcolor
\space till here
\blank
from here
  \startbar[forebar]%
  \input zapf
  \removeunwantedspaces
\stopbar
\space till here

```

from here  Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called on the screen, will make everything automatic from now on.  till here

from here 

till here

Watch how we can use the order to hide content. By default rules are drawn on top of the text.

Nice effects can be accomplished with transparencies:

```

\definecolor [tblue] [b=.5,t=.25,a=1]
\setupbars [backbar] [color=tblue]
\setupbars [forebar] [color=tblue]

```

We use as example:

```

from here {\white \backbar{test test}
  \backbar {nested nested} \backbar{also also}} till here
from here {\white \backbar{test test
  \backbar {nested nested}      also also}} till here
from here {\white \backbar{test test
  \backbar {nested nested}      also also}} till here

```

```

from here test test nested nested also also till here from here test test nested
nested also also till here from here test test nested nested also also till here

```

The darker nested variant is just the result of two transparent bars on top of each other. We can limit stacking, for instance:

```

\setupbars[backbar][max=1]
\setupbars[forebar][max=1]

```

This gives

```

from here test test nested nested also also till here from here test test nested
nested also also till here from here test test nested nested also also till here

```

There are currently some limitations that are mostly due to the fact that we use only one attribute for this feature and a change in value triggers another handling. So, we have no real nesting here.

The default commands are defined as follows:

```

\definebar[overstrike] [method=0,dy= 0.4,offset= 0.5]
\definebar[underbar]   [method=1,dy=-0.4,offset=-0.3]
\definebar[overbar]    [method=1,dy= 0.4,offset= 1.8]

\definebar[overstrikes] [overstrike] [continue=yes]
\definebar[underbars]  [underbar]    [continue=yes]
\definebar[overbars]   [overbar]     [continue=yes]

```

As the implementation is rather non-intrusive you can use bars almost everywhere. You can underbar a whole document but equally well you can stick to fooling around with for instance formulas.

```

\definecolor [tred]    [r=.5,t=.25,a=1]
\definecolor [tgreen] [g=.5,t=.25,a=1]
\definecolor [tblue]  [b=.5,t=.25,a=1]

```

```

\definebar [mathred] [backbar] [color=tred]
\definebar [mathgreen] [backbar] [color=tgreen]
\definebar [mathblue] [backbar] [color=tblue]

\startformula
  \mathred{e} = \mathgreen{\white mc} ^ {\mathblue{\white e}}
\stopformula

```

We get:

$$e = e$$

We started this chapter with some words on grouping. In the examples you see no difference between adding bars and for instance applying color. However you need to keep in mind that this is only because behind the screens we keep the current settings along with the attribute. In practice this is only noticeable when you do lots of (local) changes to the settings. Take:

```
{test test test \setupbars[color=red] \underbar{test} test}
```

This results in a local change in settings, which in turn will associate a new attribute to `\underbar`. So, in fact the following underbar becomes a different one than previous underbars. When the page is prepared, the unique attribute value will relate to those settings. Of course there are more mechanisms where such associations take place.

3.4 More to come

Is this all there is? No, as usual the underlying mechanisms can be used for other purposes as well. Take for instance inline notes:

```

According to the wikipedia this is the longest English word:
pneumonoultramicroscopicsilicovolcanoconiosis~\shiftup {other long
words are pseudopseudohypoparathyroidism and
flocci-nauci-nihili-pili-fication}. Of course in languags like Dutch and
German we can make arbitrary long words by pasting words together.

```

This will produce:

According to the wikipedia this is the longest English word: pneumonoultra-
microscopicsilicovolcanoconiosis other long words are pseudopseudohypoparathyroidism and flocci-
naucinihilipilification. Of course in languags like Dutch and German we can make
arbitrary long words by pasting words together.

I wonder when users really start using such features.

3.5 Summary

Although under the hood the MkIV bar commands are quite different from their MkII counterparts users probably won't notice much difference at first sight. However, the new implementation does not interfere with the par builder and other mechanisms. Plus, it is configurable and it offers more functionality. However, as it is processed rather delayed, side effects might occur that are not foreseen.

So, if you ever notice such unexpected side effects, you know where it might result from: what you asked for is processed much later and by then the circumstances might have changed. If you suspect that it relates to grouping there is a simple remedy: define a new bar command in the document preamble instead of changing properties mid-document. After all, you are supposed to separate rendering and content in the first place.

4 The font name mess

4.1 Introduction

When $\text{T}_{\text{E}}\text{X}$ came around it shipped with its own fonts. At that moment the $\text{T}_{\text{E}}\text{X}$ font universe was a small and well known territory. The 'only' hassle was that one needed to make sure that the right kind of bitmap was available for the printer.

When other languages than English came into the picture things became more complex as now fonts instances in specific encodings showed up. After a couple of years the by then standardised $\text{T}_{\text{E}}\text{X}$ distributions carried tens of thousands of font files. The reason for this was simple: $\text{T}_{\text{E}}\text{X}$ fonts could only have 256 characters and therefore there were quite some encodings. Also, large cjk fonts could easily have hundreds of metric files per font. Distributions also provide metrics for commercial fonts although I could never use them and as a result have many extra metric files in my personal trees (generated by $\text{T}_{\text{E}}\text{Xfont}$).⁷

At the input side many problems related to encodings were solved by Unicode. So, when the more Unicode aware fonts showed up, it looked like things would become easier. For instance, no longer were choices for encodings needed. Instead one had to choose features and enable languages and scripts and so the problem of the multitude of files was replaced by the necessity to know what some font actually provides. But still, for the average user it can be seen as an improvement.

A rather persistent problem remained, especially for those who want to use different fonts and or need to install fonts on the system that come from elsewhere (either free or commercial): the names used for fonts. You may argue that modern $\text{T}_{\text{E}}\text{X}$ engines and macro packages can make things easier, especially as one can call up fonts by their names instead of their filenames, but actually the problem has worsened. With traditional $\text{T}_{\text{E}}\text{X}$ you definitely get an error when you mistype a filename or call for a font that is not on your system. The more modern $\text{T}_{\text{E}}\text{X}$'s macro packages can provide fallback mechanisms and you can end up with something you didn't ask for.

For years one of the good things of $\text{T}_{\text{E}}\text{X}$ was its stability. If we forget about changes in content, macro packages and/or hyphenation patterns, documents could render more or less the same for years. This is because fonts didn't change. However, now that fonts are more complex, bugs gets fixed and thereby results can differ. Or, if you use platform fonts, your updated operating system might have new or even different variants. Or, if you access your fonts by

⁷ Distributions like $\text{T}_{\text{E}}\text{XLive}$ have between 50.000 and 100.000 files, but derivatives like the $\text{ConT}_{\text{E}}\text{Xt}$ minimal are much smaller.

fontname, a lookup can resolve differently.

The main reason for this is that fontnames as well as filenames of fonts are highly inconsistent across vendors, within vendors and platforms. As we have to deal with this matter, in MkIV we have several ways to address a font: by filename, by fontname, and by specification. In the next sections I will describe all three.

4.2 Method 1: file

The most robust way to specify what fonts is to be used is the filename. This is done as follows:

```
\definefont[SomeFont][file:lmmono10-regular]
```

A filename lookup is case insensitive and the name you pass is exact. Of course the file: prefix (as with any prefix) can be used in font synonyms as well. You may add a suffix, so this is also valid:

```
\definefont[SomeFont][file:lmmono10-regular.otf]
```

By default ConTeXt will first look for an OpenType font so in both cases you will get such a font. But how do you know what the filename is? You can for instance check it out with:

```
mtxrun --script font --list --file --pattern="lm*mono"
```

This reports some information about the file, like the weight, style, width, fontname, filename and optionally the subfont id and a mismatch between the analysed weight and the one mentioned by the font.

latinmodernmonolight	light	normal	normal	lmmonolt10regular	lmmonolt10-regular.otf
latinmodernmonoproplight	light	italic	normal	lmmonoprolt10oblique	lmmonoprolt10-oblique.otf
latinmodernmono	normal	normal	normal	lmmono9regular	lmmono9-regular.otf
latinmodernmonoprop	normal	italic	normal	lmmonoprop10oblique	lmmonoprop10-oblique.otf
latinmodernmono	normal	italic	normal	lmmono10italic	lmmono10-italic.otf
latinmodernmono	normal	normal	normal	lmmono8regular	lmmono8-regular.otf
latinmodernmonolightcond	light	italic	condensed	lmmonoltcond10oblique	lmmonoltcond10-oblique.otf
latinmodernmonolight	light	italic	normal	lmmonolt10oblique	lmmonolt10-oblique.otf
latinmodernmonolightcond	light	normal	condensed	lmmonoltcond10regular	lmmonoltcond10-regular.otf
latinmodernmonolight	bold	italic	normal	lmmonolt10boldoblique	lmmonolt10-boldoblique.otf
latinmodernmonocaps	normal	italic	normal	lmmonocaps10oblique	lmmonocaps10-oblique.otf
latinmodernmonoproplight	bold	italic	normal	lmmonoprolt10boldoblique	lmmonoprolt10-boldoblique.otf
latinmodernmonolight	bold	normal	normal	lmmonolt10bold	lmmonolt10-bold.otf
latinmodernmonoproplight	bold	normal	normal	lmmonoprolt10bold	lmmonoprolt10-bold.otf

latinmodernmonoslant	normal	normal	normal	lmmonoslant10regular	lmmonoslant10-regular.otf
latinmodernmono	normal	normal	normal	lmmono12regular	lmmono12-regular.otf
latinmodernmonocaps	normal	normal	normal	lmmonocaps10regular	lmmonocaps10-regular.otf
latinmodernmonoprop	normal	normal	normal	lmmonoprop10regular	lmmonoprop10-regular.otf
latinmodernmono	normal	normal	normal	lmmono10regular	lmmono10-regular.otf
latinmodernmonoproplight	light	normal	normal	lmmonoproplt10regular	lmmonoproplt10-regular.otf

4.3 Method 1: name

Instead of lookup by file, you can also use names. In the font database we store references to the fontname and fullname as well as some composed names from information that comes with the font. This permits rather liberal naming and the main reason is that we can more easily look up fonts. In practice you will use names that are as close to the filename as possible.

```
mtxrun --script font --list --name --pattern="lmmono*regular" --all
```

This gives on my machine:

```
lmmono10regular      lmmono10regular      lmmono10-regular.otf
lmmono12regular      lmmono12regular      lmmono12-regular.otf
lmmono8regular       lmmono8regular       lmmono8-regular.otf
lmmono9regular       lmmono9regular       lmmono9-regular.otf
lmmonocaps10regular  lmmonocaps10regular  lmmonocaps10-regular.otf
lmmonolt10regular    lmmonolt10regular    lmmonolt10-regular.otf
lmmonoltcond10regular  lmmonoltcond10regular  lmmonoltcond10-regular.otf
lmmonoprop10regular  lmmonoprop10regular  lmmonoprop10-regular.otf
lmmonoproplt10regular  lmmonoproplt10regular  lmmonoproplt10-regular.otf
lmmonoslant10regular  lmmonoslant10regular  lmmonoslant10-regular.otf
```

It does not show from this list but with name lookups first OpenType fonts are checked and then Type1. In this case there are Type1 variants as well but they are ignored. Fonts are registered under all names that make sense and can be derived from its description. So:

```
mtxrun --script font --list --name --pattern="latinmodern*mono" --all
```

will give:

```
latinmodernmono      lmmono9regular      lmmono9-regular.otf
latinmodernmonocaps  lmmonocaps10oblique  lmmonocaps10-oblique.otf
latinmodernmonocapsitalic  lmmonocaps10oblique  lmmonocaps10-oblique.otf
latinmodernmonocapsnormal  lmmonocaps10oblique  lmmonocaps10-oblique.otf
latinmodernmonolight  lmmonolt10regular    lmmonolt10-regular.otf
latinmodernmonolightbold  lmmonolt10boldoblique  lmmonolt10-boldoblique.otf
latinmodernmonolightbolditalic  lmmonolt10boldoblique  lmmonolt10-boldoblique.otf
latinmodernmonolightcond  lmmonoltcond10oblique  lmmonoltcond10-oblique.otf
```

latinmodernmonolightconditalic	lmmnoltcond10oblique	lmmnoltcond10-oblique.otf
latinmodernmonolightcondlight	lmmnoltcond10oblique	lmmnoltcond10-oblique.otf
latinmodernmonolightitalic	lmmnolt10oblique	lmmnolt10-oblique.otf
latinmodernmonolightlight	lmmnolt10regular	lmmnolt10-regular.otf
latinmodernmononormal	lmmno9regular	lmmno9-regular.otf
latinmodernmonoprop	lmmnopropl10oblique	lmmnopropl10-oblique.otf
latinmodernmonopropitalic	lmmnopropl10oblique	lmmnopropl10-oblique.otf
latinmodernmonopropplight	lmmnoproplt10oblique	lmmnoproplt10-oblique.otf
latinmodernmonopropplightbold	lmmnoproplt10boldoblique	lmmnoproplt10-boldoblique.otf
latinmodernmonopropplightbolditalic	lmmnoproplt10boldoblique	lmmnoproplt10-boldoblique.otf
latinmodernmonopropplightitalic	lmmnoproplt10oblique	lmmnoproplt10-oblique.otf
latinmodernmonopropplightlight	lmmnoproplt10oblique	lmmnoproplt10-oblique.otf
latinmodernmonopropnormal	lmmnopropl10oblique	lmmnopropl10-oblique.otf
latinmodernmonoslantend	lmmnoslant10regular	lmmnoslant10-regular.otf
latinmodernmonoslantendnormal	lmmnoslant10regular	lmmnoslant10-regular.otf

Watch the 9 point version in this list. It happens that there are 9, 10 and 12 point regular variants but all those extras come in 10 point only. So we get a mix and if you want a specific design size you really have to be more specific. Because one font can be registered with its fontname, fullname etc. it can show up more than once in the list. You get what you ask for.

With this obscurity you might wonder why names make sense as lookups. One advantage is that you can forget about special characters. Also, Latin Modern with its design sizes is probably the worst case. So, although for most fonts a name like the following will work, for Latin Modern it gives one of the design sizes:

```
\definefont[SomeFont][name:latinmodernmonolightbolditalic]
```

But this is quite okay:

```
\definefont[SomeFont][name:lmmnolt10boldoblique]
```

So, in practice this method will work out as well as the file method but you can best check if you get what you want.

4.4 Method 1: spec

We have now arrived at the third method, selecting by means of a specification. This time we take the familyname as starting point (although we have some fallback mechanisms):

```
\definefont[SomeSerif] [spec:times]
\definefont[SomeSerifBold] [spec:times-bold]
\definefont[SomeSerifItalic] [spec:times-italic]
```

```
\definefont[SomeSerifBoldItalic][spec:times-bold-italic]
```

The patterns are of the form:

```
spec:name-weight-style-width  
spec:name-weight-style  
spec:name-style
```

When only the name is used, it actually boils down to:

```
spec:name-normal-normal-normal
```

So, this is also valid:

```
spec:name-normal-italic-normal  
spec:name-normal-normal-condensed
```

Again we can consult the database:

```
mtxrun --script font --list --spec lmmmono-normal-italic
```

This prints the following list. The first column is the familyname, the fifth column the fontname:

latinmodernmono	normal	italic	normal	lmmmono10italic	lmmmono10-italic.otf
latinmodernmonoprop	normal	italic	normal	lmmmonoprop10oblique	lmmmonoprop10-oblique.otf
lmmmono10	normal	italic	normal	lmmmono10italic	lmtti10.afm
lmmmonoprop10	normal	italic	normal	lmmmonoprop10oblique	lmttto10.afm
lmmmonocaps10	normal	italic	normal	lmmmonocaps10oblique	lmtcso10.afm
latinmodernmonocaps	normal	italic	normal	lmmmonocaps10oblique	lmmmonocaps10-oblique.otf

Watch the OpenType and Type1 mix. As we're just investigating here, the lookup looks at the fontname and not at the familyname. At the \TeX end you use the familyname:

```
\definefont[SomeFont][spec:latinmodernmono-normal-italic-normal]
```

So, we have the following ways to access this font:

```
\definefont[SomeFont][file:lmmmono10-italic]  
\definefont[SomeFont][file:lmmmono10-italic.otf]  
\definefont[SomeFont][name:lmmmono10italic]  
\definefont[SomeFont][spec:latinmodernmono-normal-italic-normal]
```

As OpenType fonts are preferred over Type1 there is not much chance of a mixup.

As mentioned in the introduction, qualifications are somewhat inconsistent. Among the weight we find: black, bol, bold, demi, demibold, extrabold, heavy, light, medium, mediumbold, regular, semi, semibold, ultra, ultrabold and ultralight. Styles are: ita, ital, italic, roman, regular, reverseoblique, oblique and slanted. Examples of width are: book, cond, condensed, expanded, normal and thin. Finally we have alternatives which can be anything.

When doing a lookup, some normalizations takes place, with the default always being 'normal'. But still the repertoire is large:

helveticaneue medium	normal normal	helveticaneuemedium	HelveticaNeue.ttc index: 0
helveticaneue bold	normal condensed	helveticaneuecondensedbold	HelveticaNeue.ttc index: 1
helveticaneue black	normal condensed	helveticaneuecondensedblack	HelveticaNeue.ttc index: 2
helveticaneue ultralight	italic thin	helveticaneueultralightitalic	HelveticaNeue.ttc index: 3
helveticaneue ultralight	normal thin	helveticaneueultralight	HelveticaNeue.ttc index: 4
helveticaneue light	italic normal	helveticaneueightitalic	HelveticaNeue.ttc index: 5
helveticaneue light	normal normal	helveticaneuelight	HelveticaNeue.ttc index: 6
helveticaneue bold	italic normal	helveticaneuebolditalic	HelveticaNeue.ttc index: 7
helveticaneue normal	italic normal	helveticaneueitalic	HelveticaNeue.ttc index: 8
helveticaneue bold	normal normal	helveticaneuebold	HelveticaNeue.ttc index: 9
helveticaneue normal	normal normal	helveticaneue	HelveticaNeue.ttc index: 10
helveticaneue normal	normal condensed	helveticaneuecondensed	hlc____.afm conflict: roman
helveticaneue bold	normal condensed	helveticaneueboldcond	hlbc____.afm
helveticaneue black	normal normal	helveticaneueblackcond	hlzc____.afm conflict: normal
helveticaneue black	normal normal	helveticaneueblack	hlbl____.afm conflict: normal
helveticaneue normal	normal normal	helveticaneueroman	lt_50259.afm conflict: regular

4.5 The font database

In MkIV we use a rather extensive font database which in addition to bare information also contains a couple of hashes. When you use ConT_EXt MkIV and install a new font, you have to regenerate the file database. In a next T_EX run this will trigger a reload of the font database. Of course you can also force a reload with:

```
mtxrun --script font --reload
```

As a summary we mention a few of the discussed calls of this script:

```
mtxrun --script font --list somename (== --pattern=*somename*)
```

```
mtxrun --script font --list --name somename
```

```
mtxrun --script font --list --name --pattern=*somename*
```

```

mtxrun --script font --list --spec somename
mtxrun --script font --list --spec somename-bold-italic
mtxrun --script font --list --spec --pattern=*somename*
mtxrun --script font --list --spec --filter="fontname=somename"
mtxrun --script font --list --spec --filter="familyname=somename,weight=bold,style=italic,width=condensed"

mtxrun --script font --list --file somename
mtxrun --script font --list --file --pattern=*somename*

```

The lists shown in before depend on what fonts are installed and their version. They might not reflect reality at the time you read this.

4.6 Interfacing

Regular users never deal with the font database directly. However, if you write font loading macros yourself, you can access the database from the \TeX end. First we show an example of an entry in the database, in this case TeXGyreTermes Regular.

```

{
  designsize = 100,
  familyname = "texgyretermes",
  filename = "texgyretermes-regular.otf",
  fontname = "texgyretermesregular",
  fontweight = "regular",
  format = "otf",
  fullname = "texgyretermesregular",
  maxsize = 200,
  minsize = 50,
  rawname = "TeXGyreTermes-Regular",
  style = "normal",
  variant = "",
  weight = "normal",
  width = "normal",
}

```

Another example is Helvetica Neue Italic:

```

{
  designsize = 0,
  familyname = "helveticaneue",
  filename = "HelveticaNeue.ttc",
  fontname = "helveticaneueitalic",
  fontweight = "book",
  format = "ttc",
}

```

```

    fullname = "helveticaneueitalic",
    maxsize = 0,
    minsize = 0,
    rawname = "Helvetica Neue Italic",
    style = "italic",
    subfont = 8,
    variant = "",
    weight = "normal",
    width = "normal",
}

```

As you can see, some fields can be meaningless, like the sizes. As using the low level T_EX interface assumes some knowledge, we stick here to an example:

```

\def\TestLookup#1%
  {\dlookupfontbyspec{#1}
   pattern: #1, found: \dlookupnofound
   \blank
   \dorecurese {\dlookupnofound} {%
    \recurselevel:~\dlookupgetkeyofindex{fontname}{\recurselevel}%
    \quad
   }%
   \blank}

\TestLookup{familyname=helveticaneue}
\TestLookup{familyname=helveticaneue,weight=bold}
\TestLookup{familyname=helveticaneue,weight=bold,style=italic}

```

You can use the following commands:

```

\dlookupfontbyspec    {key=value list}
\dlookupnofound
\dlookupgetkeyofindex {key}{index}
\dlookupgetkey        {key}

```

First you do a lookup. After that there can be one or more matches and you can access the fields of each match. What you do with the information is up to yourself.

4.7 A few remarks

The fact that modern T_EX engines can access system fonts is promoted as a virtue. The previous sections demonstrated that in practice this does not really

free us from a name mess. Of course, when we use a really small T_EX tree, and system fonts only, there is not much that can go wrong, but when you have extra fonts installed there can be clashes.

We're better off with filenames than we were in former times when operating systems and media forced distributors to stick to 8 characters in filenames. But that does not guarantee that today's shipments are more consistent. And as there are still some limitations in the length of fontnames, obscure names will be with us for a long time to come.

5 Deeply nested notes

5.1 Introduction

One of the mechanisms that is not on a users retina when he or she starts using \TeX is ‘inserts’. An insert is material that is entered at one point but will appear somewhere else in the output. Footnotes for instance can be implemented using inserts. You create a reference symbol in the running text and put note text at the bottom of the page or at the end of a chapter or document. But as you don’t want to do that moving around of notes yourself \TeX provides macro writers with the inserts mechanism that will do some of the housekeeping. Inserts are quite clever in the sense that they are taken into account when \TeX splits off a page. A single insert can even be split over two or more pages.

Other examples of inserts are floats that move to the top or bottom of the page depending on requirements and/or available space. Of course the macro package is responsible for packaging such a float (for instance an image) but by finally putting it in an insert \TeX itself will attempt to deal with accumulated floats and help you move kept over floats to following pages. When the page is finally assembled (in the output routine) the inserts for that page become available and can be put at the spot where they belong. In the process \TeX has made sure that we have the right amount of space available.

However, let’s get back to notes. In $\text{Con}\TeX\text{t}$ we can have many variants of them, each taken care of by its own class of inserts. This works quite well, as long as a note is visible for \TeX which means as much as: ends up in the main page flow. Consider the following situation:

```
before \footnote{the note} after
```

When the text is typeset, a symbol is placed directly after before and the note itself ends up at the bottom of the page. It also works when we wrap the text in an horizontal box:

```
\hbox{before \footnote{the note} after}
```

But it fails as soon as we go further:

```
\hbox{\hbox{before \footnote{the note} after}}
```

Here we get the reference but no note. This also fails:

```
\vbox{before \footnote{the note} after}
```


Can you imagine what happens if we do the following?

```
\starttabulate
\NC knuth \NC test \footnote{knuth} \input knuth \NC \NR
\NC tufte \NC test \footnote{tufte} \input tufte \NC \NR
\NC Ward \NC test \footnote{ward} \input ward \NC \NR
\stoptabulate
```

This mechanism uses alignments as well as quite some boxes. The paragraphs are nicely split over pages but still appear as boxes to $\text{T}_{\text{E}}\text{X}$ which make inserts invisible. Only the three symbols would remain visible. But because in $\text{ConT}_{\text{E}}\text{Xt}$ we know when notes tend to disappear, we take some provisions, and contrary to what you might expect the notes actually do show up. However, they are flushed in such a way that they end up on the page where the table ends. Normally this is no big deal as we will often use local notes that end up at the end of the table instead of the bottom of the page, but still.

The mechanism to deal with notes in $\text{ConT}_{\text{E}}\text{Xt}$ is somewhat complex at the source code level. To mention a few properties we have to deal with:

- Notes are collected and can be accessed any time.
- Notes are flushed either directly or delayed.
- Notes can be placed anywhere, any time, perhaps in subsets.
- Notes can be associated to lines in paragraphs.
- Notes can be placed several times with different layouts.

So, we have some control over flushing and placement, but real synchronization between for instance table entries having notes and the note content ending up on the same page is impossible.

In the $\text{LuaT}_{\text{E}}\text{X}$ team we have been discussing more control over inserts and we will definitely deal with that in upcoming releases as more control is needed for complex multi-column document layouts. But as we have some other priorities these extensions have to wait.

As a prelude to them I experimented a bit with making these deeply buried inserts visible. Of course I use Lua for this as $\text{T}_{\text{E}}\text{X}$ itself does not provide the kind of access we need for this kind of manipulations.

5.2 Deep down inside

Say that we have the following boxed footnote. How does that end up in $\text{LuaT}_{\text{E}}\text{X}$?

`\vbox{a\footnote{b}c}`

Actually it depends on the macro package but the principles remain the same. In Lua \TeX 0.50 and the Con \TeX t version used at the time of this writing we get (nested) linked list that prints as follows:

```
<node 26 < 862 > nil : vlist 0>
  <node 401 < 838 > 507 : hlist 1>
    <node 30 < 611 > 580 : whatsit 6>
    <node 611 < 580 > 493 : hlist 0>
    <node 580 < 493 > 653 : glyph 256>
    <node 493 < 653 > 797 : penalty 0>
    <node 653 < 797 > 424 : kern 1>
    <node 797 < 424 > 826 : hlist 2>
      <node 445 < 563 > nil : hlist 2>
        <node 420 < 817 > 821 : whatsit 35>
        <node 817 < 821 > nil : glyph 256>
      <node 507 < 826 > 1272 : kern 1>
      <node 826 < 1272 > 1333 : glyph 256>
      <node 1272 < 1333 > 830 : penalty 0>
      <node 1333 < 830 > 888 : glue 15>
      <node 830 < 888 > nil : glue 9>
    <node 838 < 507 > nil : ins 131>
```

The numbers are internal references to the node memory pool. Each line represents a node:

```
<node prev_index < index > next_index : type subtype>
```

The whatsits carry directional information and the deeply nested hlist is the note symbol. If we forget about whatsits, kerns and penalties, we can simplify this listing to:

```
<node 26 < 862 > nil : vlist 0>
  <node 401 < 838 > 507 : hlist 1>
    <node 580 < 493 > 653 : glyph 256>
    <node 797 < 424 > 826 : hlist 2>
      <node 445 < 563 > nil : hlist 2>
        <node 817 < 821 > nil : glyph 256>
      <node 826 < 1272 > 1333 : glyph 256>
    <node 838 < 507 > nil : ins 131>
```

So, we have a vlist (the `\vbox`), which has one line being a hlist. Inside we have a glyph (the ‘a’) followed by the raised symbol (the ‘¹’) and next comes the

second glyph (the 'b'). But watch how the insert ends up at the end of the line. Although the insert will not show up in the document, it sits there waiting to be used. So we have:

```
<node 26 < 862 > nil : vlist 0>
  <node 401 < 838 > 507 : hlist 1>
    <node 838 < 507 > nil : ins 131>
```

but we need:

```
<node 26 < 862 > nil : vlist 0>
  <node 401 < 838 > 507 : hlist 1>
<node 838 < 507 > nil : ins 131>
```

Now, we could use the fact that inserts end up at the end of the line, but as we need to recursively identify them anyway, we cannot actually use this fact to optimize the code.

In case you wonder how multiple inserts look like, here is an example:

```
\vbox{a\footnote{b}\footnote{c}d}
```

This boils down to:

```
<node 26 < 1324 > nil : vlist 0>
  <node 401 < 1348 > 507 : hlist 1>
    <node 1348 < 507 > 457 : ins 131>
    <node 507 < 457 > nil : ins 131>
```

In case you wonder what more can end up at the end, vertically adjusted material (`\vadjust`) as well as marks (`\mark`) also gets that treatment.

```
\vbox{a\footnote{b}\vadjust{c}\footnote{d}e\mark{f}}
```

As you see, we start with the line itself, followed by a mixture of inserts and vertical adjusted content (that will be placed before that line). This trace also shows the list 2 levels deep.

```
<node 26 < 1324 > nil : vlist 0>
  <node 401 < 1348 > 507 : hlist 1>
    <node 1348 < 507 > 862 : ins 131>
    <node 507 < 862 > 240 : hlist 1>
    <node 862 < 240 > 2288 : ins 131>
    <node 240 < 2288 > nil : mark 0>
```

Currently `vadjust` nodes have the same subtype as an ordinary `hlist` but in Lua_T_E_X versions beyond 0.50 they will have a dedicated subtype.

We can summarize the pattern of one ‘line’ in a vertical list as:

```
[hlist][insert|mark|vadjust]*[penalty|glue]+
```

In case you wonder what happens with for instance specials, literals (and other whatits): these end up in the `hlist` that holds the line. Only inserts, marks and `vadjusts` migrate to the outer level, but as they stay inside the `vlist`, they are not visible to the page builder unless we're dealing with the main vertical list. Compare:

```
this is a regular paragraph possibly with inserts and they
will be visible as the lines are appended to the main
vertical list \par
```

with:

```
but \vbox {this is a nested paragraph where inserts will
stay with the box} and not migrate here \par
```

So much for the details; let's move on the how we can get around this phenomenon.

5.3 Some Lua_T_E_X magic

The following code is just the first variant I made and Con_T_E_Xt ships with a more extensive variant. Also, in Con_T_E_Xt this is part of a larger suite of manipulative actions but it does not make much sense (at least not now) to discuss this framework here.

We start with defining a couple of convenient shortcuts.

```
local hlist = node.id('hlist')
local vlist = node.id('vlist')
local ins   = node.id('ins')
```

We can write a more compact solution but splitting up the functionality better shows what we're doing. The main migration function hooks into the callback `build_page`. Contrary to other callbacks that do phases in building lists and pages this callback does not expect the head of a list as argument. Instead, we operate directly on the additions to the main vertical list which is accessible as `tex.lists.contrib_head`.

```

local deal_with_inserts -- forward reference

local function migrate_inserts(when)
  local current = tex.lists.contrib_head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      current = deal_with_inserts(current)
    end
    current = current.next
  end
end

callback.register('buildpage_filter',migrate_inserts)

```

So, effectively we scan for vertical and horizontal lists and deal with embedded inserts when we find them. In ConTEXt the migratory function is just one of the functions that is applied to this filter.

We locate inserts and collect them in a list with first and last as head and tail and do so recursively. When we have run into inserts we insert them after the horizontal or vertical list that had embedded them.

```

local locate -- forward reference

deal_with_inserts = function(head)
  local h, first, last = head.list, nil, nil
  while h do
    local id = h.id
    if id == vlist or id == hlist then
      h, first, last = locate(h,first,last)
    end
    h = h.next
  end
  if first then
    local n = head.next
    head.next = first
    first.prev = head
    if n then
      last.next = n
      n.prev = last
    end
  end
  return last
end

```

```

    else
        return head
    end
end

```

The locate function removes inserts and adds them to a new list, that is passed on down in recursive calls and eventually is returned back to the caller.

```

locate = function(head,first,last)
    local current = head
    while current do
        local id = current.id
        if id == vlist or id == hlist then
            current.list, first, last = locate(current.list,first,last)
            current = current.next
        elseif id == ins then
            local insert = current
            head, current = node.remove(head,current)
            insert.next = nil
            if first then
                insert.prev = last
                last.next = insert
            else
                insert.prev = nil
                first = insert
            end
            last = insert
        else
            current = current.next
        end
    end
    return head, first, last
end

```

As we can encounter the content several times in a row, it makes sense to mark already processed inserts. This can for instance be done by setting an attribute. Of course one has to make sure that this attribute is not used elsewhere.

```

if not node.has_attribute(current,8061) then
    node.set_attribute(current,8061,1)
    current = deal_with_inserts(current)
end

```

or integrated:

```
local has_attribute = node.has_attribute
local set_attribute = node.set_attribute

local function migrate_inserts(when)
  local current = tex.lists.contrib_head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      if has_attribute(current,8061) then
        -- maybe some tracing message
      else
        set_attribute(current,8061,1)
        current = deal_with_inserts(current)
      end
    end
    current = current.next
  end
end

callback.register('buildpage_filter',migrate_inserts)
```

5.4 A few remarks

Surprisingly, the amount of code needed for insert migration is not that large. This makes one wonder why \TeX does not provide this feature itself as it could have saved macro writers quite some time and headaches. Performance can be a reason, unpredictable usage and side effects might be another. Only one person knows the answer.

In $\text{Con}\TeX$ t this mechanism is built in and it can be enabled by saying:

```
\automigrateinserts
\automigratemarks
```

As you can see here, we can also migrate marks. Future versions of $\text{Con}\TeX$ t will do this automatically and also provide some control over what classes of inserts are moved around. We will probably overhaul the note handling mechanism a few more times anyway as $\text{Lua}\TeX$ evolves and the demands from critical editions that use many kind of notes raise.

5.5 Summary of code

The following code should work in plain \TeX :

```
\directlua 0 {
local hlist      = node.id('hlist')
local vlist      = node.id('vlist')
local ins        = node.id('ins')
local has_attribute = node.has_attribute
local set_attribute = node.set_attribute

local status = 8061

local function locate(head,first,last)
  local current = head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      current.list, first, last = locate(current.list,first,last)
      current = current.next
    elseif id == ins then
      local insert = current
      head, current = node.remove(head,current)
      insert.next = nil
      if first then
        insert.prev, last.next = last, insert
      else
        insert.prev, first = nil, insert
      end
      last = insert
    else
      current = current.next
    end
  end
  return head, first, last
end

local function migrate_inserts(where)
  local current = tex.lists.contrib_head
  while current do
    local id = current.id
    if id == vlist or id == hlist and
      not has_attribute(current,status) then
      set_attribute(current,status,1)
    end
  end
end
```



```

    local h, first, last = current.list, nil, nil
    while h do
        local id = h.id
        if id == vlist or id == hlist then
            h, first, last = locate(h,first,last)
        end
        h = h.next
    end
    if first then
        local n = current.next
        if n then
            last.next, n.prev = n, last
        end
        current.next, first.prev = first, current
        current = last
    end
end
current = current.next
end
end

callback.register('buildpage_filter', migrate_inserts)
}

```

Alternatively you can put the code in a file and load that with:

```
\directlua {require "luatex-inserts.lua"}
```

A simple plain test is:

```

\ vbox{a\footnote{1}{1}b}
\ hbox{a\footnote{2}{2}b}

```

The first footnote only shows up when we have hooked our migrator into the callback. A not that bad result for 60 lines of Lua code.

6 Backend code

6.1 Introduction

In ConT_EXt we've always separated the backend code in so called driver files. This means that in the code related to typesetting only calls to the api take place, and no backend specific code is to be used. That way we can support backend like dvipsone (and dviwindo), dvips, acrobat, pdftex and dvi_{pdf}mx with one interface. A similar model is used in MkIV although at the moment we only have one backend: pdf.⁸

Some ConT_EXt users like to add their own pdf specific code to their styles or modules. However, such extensions can interfere with existing code, especially when resources are involved. This has to be done via the official helper macros.

In the next sections an overview will be given of the current approach. There are still quite some rough edges but these will be polished as soon as the backend code is more isolated in LuaT_EX itself.

6.2 Structure

A pdf file is tree of indirect objects. Each object has a number and the file contains a table (or more tables) that relates these numbers to positions in a file (or position in a compressed object stream). That way a file can be viewed without reading all data: a viewer only loads what is needed.

```
1 0 obj <<
  /Name (test) /Address 2 0 R
>>
2 0 obj [
  (Main Street) (24) (postal code) (MyPlace)
]
```

For the sake of the discussion we consider strings like (test) also to be objects. In the next table we list what we can encounter in a pdf file. There can be indirect objects in which case a reference is used (2 0 R) and direct ones.

type	form	meaning
constant	/...	A symbol (prescribed string).
string	(...)	A sequence of characters in pdfdoc encoding
unicode	<...>	A sequence of characters in utf16 encoding

⁸ At this moment we only support the native pdf backend but future versions might support xml (html) output as well.

number	3.1415	A number constant.
boolean	true/false	A boolean constant.
reference	N 0 R	A reference to an object
dictionary	<< ... >>	A collection of key value pairs where the value itself is an (indirect) object.
array	[...]	A list of objects or references to objects.
stream		A sequence of bytes either or not packaged with a dictionary that contains descriptive data.
xform		A special kind of object containing an reusable blob of data, for example an image.

While writing additional backend code, we mostly create dictionaries.

```
<< /Name (test) /Address 2 0 R >>
```

In this case the indirect object can look like:

```
[ (Main Street) (24) (postal code) (MyPlace) ]
```

It all starts in the document's root object. From there we access the page tree and resources. Each page carries its own resource information which makes random access easier. A page has a page stream and there we find the to be rendered content as a mixture of (Unicode) strings and special drawing and rendering operators. Here we will not discuss them as they are mostly generated by the engine itself or dedicated subsystems like the MetaPost converter. There we use literal or `\latelua` whatsits to inject code into the current stream.

In the ConTeXt MkII backend drivers code you will see objects in their verbose form. The content is passed on using special primitives, like `\pdfobj`, `\pdfannot`, `\pdfcatalog`, etc. In MkIV no such primitives are used. In fact, some of them are overloaded to do nothing at all. In the Lua backend code you will find function calls like:

```
local d = lpdf.dictionary {
  Name    = lpdf.string("test"),
  Address = lpdf.array {
    "Main Street", "24", "postal code", "MyPlace",
  }
}
```

Equally valid is:

```
local d = lpdf.dictionary()
d.Name = "test"
```

Eventually the object will end up in the file using calls like:

```
local r = pdf.immediateobj(tostring(d))
```

or using the wrapper (which permits tracing):

```
local r = lpdf.flushobject(d)
```

The object content will be serialized according to the formal specification so the proper << >> etc. are added. If you want the content instead you can use a function call:

```
local dict = d()
```

An example of using references is:

```
local a = lpdf.array {
    "Main Street", "24", "postal code", "MyPlace",
}
local d = lpdf.dictionary {
    Name    = lpdf.string("test"),
    Address = lpdf.reference(a),
}
local r = lpdf.flushobject(d)
```

We have the following creators. Their arguments are optional.

function	optional parameter
lpdf.stream	indexed table of operators
lpdf.dictionary	hash wit key/values
lpdf.array	indexed table of objects
lpdf.unicode	string
lpdf.string	string
lpdf.number	number
lpdf.constant	string
lpdf.null	
lpdf.boolean	boolean
lpdf.true	
lpdf.false	
lpdf.reference	string
lpdf.verbose	indexed table of strings

Flushing objects is done with:

```
lpdf.flushobject(obj)
```

Reserving object is of course possible and done with:

```
local r = lpdf.reserveobject()
```

Such an object is flushed with:

```
lpdf.flushobject(r,obj)
```

We also support named objects:

```
lpdf.reserveobject("myobject")
```

```
lpdf.flushobject("myobject",obj)
```

6.3 Resources

While Lua \TeX itself will embed all resources related to regular typesetting, MkIV has to take care of embedding those related to special tricks, like annotations, spot colors, layers, shades, transparencies, metadata, etc. If you ever took a look in the MkII spec-* files you might have gotten the impression that it quickly becomes messy. The code there is actually rather old and evolved in sync with the pdf format as well as pdf \TeX and dvi pdfmx maturing to their current state. As a result we have a dedicated object referencing model that sometimes results in multiple passed due to forward references. We could have gotten away from that with the latest versions of pdf \TeX as it provides means to reserve object numbers but it makes not much sense to do that now that MkII is frozen.

Because third party modules (like tikz) also can add resources like in MkII using an api that makes sure that no interference takes place. Think of macros like:

```
\pdfbackendsetcatalog      {key}{string}
\pdfbackendsetinfo         {key}{string}
\pdfbackendsetname         {key}{string}

\pdfbackendsetpageattribute {key}{string}
\pdfbackendsetpagesattribute {key}{string}
\pdfbackendsetpageresource  {key}{string}

\pdfbackendsettextgstate   {key}{pdfdata}
\pdfbackendsetcolorspace   {key}{pdfdata}
\pdfbackendsetpattern      {key}{pdfdata}
```

```
\pdfbackendsetshade      {key}{pdfdata}
```

One is free to use the Lua interface instead, as there one has more possibilities. The names are similar, like:

```
lpdf.addtoinfo(key,anything_valid_pdf)
```

At the time of this writing (LuaTeX .50) there are still places where TeX and Lua code is interwoven in a non optimal way, but that will change in the future as the backend is completely separated and we can do more TeX trickery at the Lua end.

Also, currently we expose more of the backend code that we like and future versions will have a more restricted access. The following function will stay public:

```
lpdf.addtopageresources (key,value)
lpdf.addtopageattributes (key,value)
lpdf.addtopagesattributes(key,value)
```

```
lpdf.adddocumenttextgstate(key,value)
lpdf.adddocumentcolorspac(key,value)
lpdf.adddocumentpattern (key,value)
lpdf.adddocumentshade (key,value)
```

```
lpdf.addtocatalog (key,value)
lpdf.addtoinfo (key,value)
lpdf.addtonames (key,value)
```

There a bit of tracing built in and we will add some more in due time:

```
\enabletrackers
  [backend.finalizers,
   backend.resources,
   backend.objects,
   backend.detail]
```

As with all trackers you can also pass them on the command line, for example:

```
context --trackers=backend.* yourfile
```

The reference related backend mechanisms have their own trackers.

6.4 Transformations

There is at the time of this writing still some backend related code at the $\text{T}_{\text{E}}\text{X}$ end that needs a cleanup. Most noticeable is the code that deals with transformations (like scaling). At some moment in $\text{pdfT}_{\text{E}}\text{X}$ a primitive was introduced but it was not completely covering the transform matrix so we never used it. In $\text{LuaT}_{\text{E}}\text{X}$ we will come up with a better mechanism. Till that moment we stick to the MkII method.

6.5 Annotations

The Lua based backend of MkIV is not so much less code, but definitely cleaner. The reason why there is quite some code is because in $\text{ConT}_{\text{E}}\text{Xt}$ we also handle annotations and destinations in Lua. In other words: $\text{T}_{\text{E}}\text{X}$ is not bothered by the backend any more. We could make that split without too much impact as we never depended on $\text{pdfT}_{\text{E}}\text{X}$ hyperlink related features and used generic annotations instead. It's for that reason that $\text{ConT}_{\text{E}}\text{Xt}$ has always been able to nest hyperlinks and have annotations with a chain of actions.

Another reason for doing it all at the Lua end is that as in MkII we have to deal with the rather hybrid cross reference mechanisms which uses a sort of language and parsing this is also easier at the Lua end. Think of:

```
\definereference[somesound][StartSound(attention)]
```

```
\at {just some page} [someplace,somesound,StartMovie(somemovie)]
```

We parse the specification expanding shortcuts when needed, create an action chain, make sure that the movie related resources are taken care of (normally the movie itself will be a figure), and turn the three words into hyperlinks. As this all happens in Lua we have less $\text{T}_{\text{E}}\text{X}$ code. Contrary to what you might expect, the Lua code is not that much faster as the MkII $\text{T}_{\text{E}}\text{X}$ code is rather optimized.

Special features like JavaScript as well as widgets (and forms) are also reimplemented. Support for JavaScript is not that complex at all, but as in $\text{ConT}_{\text{E}}\text{Xt}$ we can organize scripts in collections and have automatic inclusion of used functions, still some code is needed. As we now do this in Lua we use less $\text{T}_{\text{E}}\text{X}$ memory. Reimplementing widgets took a bit more work as I used the opportunity to remove hacks for older viewers. As support for widgets is somewhat instable in viewers quite some testing was needed, especially because we keep supporting cloned and copied fields (resulting in widget trees).

An interesting complication with widgets is that each instance can have a lot

of properties and as we want to be able to use thousands of them in one document, each with different properties, we have efficient storage in MkII and want to do the same in Lua. Most code at the \TeX end is related to passing all those options.

You could use the Lua functions that relate to annotations etc. but normally you will use the regular Con \TeX t user interface. For practical reasons, the backend code is grouped in several tables:

The backends table has subtables for each backend and currently there is only one: pdf. Each backend provides tables itself. In the codeinjections namespace we collect functions that don't interfere with the typesetting or typeset result, like inserting resources of all kind (movies, attachment, etc.), widget related functionality, and in fact everything that does not fit into the other categories. In nodeinjections we organize functions that inject literal pdf code in the nodelist which then ends up in the pdf stream: color, layers, etc. The registrations table is reserved for functions related to resources that result from node injections: spot colors, transparencies, etc. Once the backend code is finished we might come up with another organization. No matter what we end up with, the way the backends table is supposed to be organized determines the api and those who have seen the MkII backend code will recognize some of it.

6.6 Metadata

We always had the opportunity to set the information fields in a pdf but standardization forces us to add these large verbose metadata blobs. As this blob is codes in xml we use the built in xml parser to fill a template. Thanks to extensive testing and research by Peter Rolf we now have a rather complete support for pdf/x related demands. This will definitely evolve with the advance of the pdf specification. You can replace the information with your own but we suggest that you stay away from this metadata mess as far as possible.

6.7 Helpers

If you look into the `lpdf-*.lua` files you will find more functions. Some are public helpers, like:

```
lpdf.toeight(str)    returns (string)
lpdf.cleaned(str)    returns escaped string
lpdf.tosixteen(str) returns <utf16 sequence>
```

An example of another public function is:

`\pdf.sharedobj (content)`

This one flushes the object and returns the object number. Already defined objects are reused. In addition to this code driven optimization, some other optimization and reuse takes place but all that happens without user intervention.

7 Callbacks

7.1 Introduction

Callbacks are the means to extend the basic $\text{T}_{\text{E}}\text{X}$ engine's functionality in $\text{LuaT}_{\text{E}}\text{X}$ and $\text{ConT}_{\text{E}}\text{Xt MkIV}$ uses them extensively. Although the interface is still in development we see users popping in their own functionality and although there is nothing wrong with that, it can open a can of worms.

It is for this reason that from now on we protect the MkIV callbacks from being overloaded. For those who still want to add their own code some hooks are provided. Here we will address some of these issues.

7.2 Actions

There are already quite some callbacks and we use most of them. In the following list the callbacks tagged with `enabled` are used and `frozen`, the ones tagged `disabled` are blocked and never used, while the ones tagged `undefined` are yet unused.

<code>buildpage_filter</code>	<code>enabled</code>	vertical spacing etc (mvl)
<code>char_exists</code>	<code>undefined</code>	
<code>define_font</code>	<code>enabled</code>	definition of fonts (tfontable preparation)
<code>find_data_file</code>	<code>enabled</code>	find file using resolver
<code>find_enc_file</code>	<code>enabled</code>	find file using resolver
<code>find_font_file</code>	<code>enabled</code>	find file using resolver
<code>find_format_file</code>	<code>enabled</code>	find file using resolver
<code>find_image_file</code>	<code>enabled</code>	find file using resolver
<code>find_map_file</code>	<code>enabled</code>	find file using resolver
<code>find_opentype_file</code>	<code>enabled</code>	find file using resolver
<code>find_output_file</code>	<code>enabled</code>	find file using resolver
<code>find_pk_file</code>	<code>enabled</code>	find file using resolver
<code>find_read_file</code>	<code>enabled</code>	find file using resolver
<code>find_sfd_file</code>	<code>enabled</code>	find file using resolver
<code>find_truetype_file</code>	<code>enabled</code>	find file using resolver
<code>find_typel_file</code>	<code>enabled</code>	find file using resolver
<code>find_vf_file</code>	<code>enabled</code>	find file using resolver
<code>find_write_file</code>	<code>enabled</code>	find file using resolver
<code>finish_pdffile</code>	<code>enabled</code>	
<code>hpack_filter</code>	<code>enabled</code>	all kind of horizontal manipulations
<code>hyphenate</code>	<code>disabled</code>	normal hyphenation routine, called elsewhere
<code> Kerning</code>	<code>disabled</code>	normal kerning routine, called elsewhere

ligaturing	disabled	normal ligaturing routine, called elsewhere
linebreak_filter	enabled	breaking paragraphs into lines
mlist_to_hlist	enabled	preprocessing math list
open_read_file	enabled	open file for reading
post_linebreak_filter	enabled	all kind of horizontal manipulations (after par break)
pre_dump	enabled	lua related finalizers called before we dump the format
pre_linebreak_filter	enabled	all kind of horizontal manipulations (before par break)
pre_output_filter	undefined	
process_input_buffer	disabled	actions performed when reading data
process_output_buffer	disabled	actions performed when writing data
read_data_file	enabled	read file at once
read_enc_file	enabled	read file at once
read_font_file	enabled	read file at once
read_map_file	enabled	read file at once
read_opentype_file	undefined	read file at once
read_pk_file	enabled	read file at once
read_sfd_file	enabled	read file at once
read_truetype_file	undefined	read file at once
read_type1_file	undefined	read file at once
read_vf_file	enabled	read file at once
show_error_hook	enabled	
start_page_number	enabled	actions performed at the beginning of a shipout
start_run	enabled	actions performed at the beginning of a run
stop_page_number	enabled	actions performed at the end of a shipout
stop_run	enabled	actions performed at the end of a run
token_filter	undefined	
vpack_filter	enabled	vertical spacing etc

You can be rather sure that we will eventually use all callbacks one way or the other. Also, some callbacks are only set when certain functionality is enabled.

It may sound somewhat harsh but if users kick in their own code, we cannot guarantee other ConTeXt behaviour and support becomes a pain. If you really need to use a callback yourself, you should use one of the hooks and make sure that you return the right values.

The exact working of of the callback handler is not something we need to bother users with so we stick to a simple description. The next list is not definitive and

evolves. For instance we might at some point decide to add more granularity.

We only open up some of the node list related callbacks. All callbacks related to file handling, font definition and housekeeping are frozen. Most if the mechanisms that use these callbacks have hooks anyway.

Of course you can overload the built in functionality as this is currently not protected, but we might do that as well once MkIV is stable enough. After all, at the time of this writing overloading can be handy when testing.

This leaves the node list manipulators. The are grouped as follows:

category	callback	usage
processors	<code>pre_linebreak_filter</code>	called just before the paragraph is broken into lines
	<code>hpack_filter</code>	called just before a horizontal box is constructed
finalizers	<code>post_linebreak_filter</code>	called just after the paragraph has been broken into lines
shipouts	no callback yet	applied to the box (or xform) that is to be shipped out
mvlbuilders	<code>buildpage_filter</code>	called after some material has been added to the main vertical list
vboxbuilders	<code>vpack_filter</code>	called when some material is added to a vertical box
math	<code>mlist_to_hlist</code>	called just after the math list is created, before it is turned into an horizontal list

Each category has several subcategories but for users only two make sense: before and after. Say that you want to hook some tracing into the mvlbuilder. This is how it's done:

```
function third.mymodule.myfunction(where)
  nodes.show_simple_list(tex.lists.contrib_head)
end

nodes.tasks.appendaction("processors", "before", "third.mymodule.myfunction")
```

As you can see, in this case the function gets no head passed (at least not currently). This example also assumes that you know how to access the right items. The arguments and return values are given below.⁹

⁹ This interface might change a bit in future versions of ConTeXt. Therefore we will not discuss

category	arguments	return value
processors	head, ...	head, done
finalizers	head, ...	head, done
shipouts	head	head, done
mvlbuilders		done
vboxbuilders	head, ...	head, done
math	head, ...	head, done

7.3 Tasks

In the previous section we already saw that the actions are in fact tasks and that we can append (and therefore also prepend) to a list of tasks. The before and after task lists are valid hooks for users contrary to the other tasks that can make up an action. However, the task builder is generic enough for users to be used for individual tasks that are plugged into the user hooks.

Of course at some point, too many nested tasks bring a performance penalty with them. At the end of a run MkIV reports some statistics and timings and these can give you an idea how much time is spent in Lua. Of course this is a rough estimate only.

The following tables list all the registered tasks for the processors actions:

category	function
before	unset
normalizers	fonts.collections.process fonts.checkers.missing
characters	typesetters.directions.handler typesetters.cases.handler typesetters.breakpoints.handler scripts.preprocess
words	builders.kernel.hyphenation languages.words.check
fonts	builders.paragraphs.solutions.splitters.split nodes.handlers.characters nodes.injections.handler nodes.handlers.protectglyphs builders.kernel.ligaturing builders.kernel.kerning nodes.handlers.stripping

the few more optional arguments that are possible.

	fonts.goodies.colorschemes.coloring
lists	typesetters.spacings.handler typesetters.kerns.handler typesetters.digits.handler
after	unset

Some of these do have subtasks and some of these even more, so you can imagine that quite some action is going on there.

The finalizer tasks are:

category	function
before	unset
normalizers	unset
fonts	builders.paragraphs.solutions.splitters.optimize
lists	nodes.handlers.graphicvadjust
after	unset

Shipouts concern:

category	function
before	unset
normalizers	nodes.handlers.cleanuppage nodes.references.handler nodes.destinations.handler nodes.rules.handler nodes.shifts.handler structures.tags.handler nodes.handlers.accessibility nodes.handlers.backgrounds
finishers	attributes.colors.handler attributes.transparencies.handler attributes.colorintents.handler attributes.effects.handler attributes.viewerlayers.handler
after	unset

There are not that many mvlbuilder task currently:

category	function
-----------------	-----------------

before	unset
normalizers	streams.collect nodes.handlers.migrate builders.vspacing.pagehandler
after	unset

The vboxbuilder perform similar tasks:

category	function
before	unset
normalizers	builders.vspacing.vboxhandler
after	unset

Finally, we have tasks related to the math list:

category	function
before	unset
normalizers	noads.handlers.relocate noads.handlers.resize noads.handlers.respace noads.handlers.check noads.handlers.tags
builders	builders.kernel.mlist_to_hlist
after	unset

As MkIV is developed in sync with LuaTeX and code changes from experimental to more final and reverse, you should not be too surprised if the registered function names change.

You can create your own task list with:

```
nodes.tasks.new("mytasks",{ "one", "two" })
```

After that you can register functions. You can append as well as prepend them either or not at a specific position.

```
nodes.tasks.appendaction ("mytask", "one", "bla.alpha")
nodes.tasks.appendaction ("mytask", "one", "bla.beta")
```

```
nodes.tasks.prependaction("mytask", "two", "bla.gamma")
nodes.tasks.prependaction("mytask", "two", "bla.delta")
```

```
nodes.tasks.appendaction ("mytask", "one", "bla.whatever", "bla.alpha")
```

Functions can also be removed:

```
nodes.tasks.removeaction("mytask", "one", "bla.whatever")
```

As removal is somewhat drastic, it is also possible to enable and disable functions. From the fact that with these two functions you don't specify a category (like one or two) you can conclude that the function names need to be unique within the task list or else all with the same name within this task will be disabled.

```
nodes.tasks.enableaction ("mytask", "bla.whatever")
nodes.tasks.disableaction("mytask", "bla.whatever")
```

The same can be done with a complete category:

```
nodes.tasks.enablegroup ("mytask", "one")
nodes.tasks.disablegroup("mytask", "one")
```

There is one function left:

```
nodes.tasks.actions("mytask", 2)
```

This function returns a function that when called will perform the tasks. In this case the function takes two extra arguments in addition to head.¹⁰

Tasks themselves are implemented on top of sequences but we won't discuss them here.

7.4 Paragraph and page builders

Building paragraphs and pages is implemented differently and has no user hooks. There is a mechanism for plugins but the interface is quite experimental.

¹⁰ Specifying this number permits for some optimization but is not really needed

8 Bibliographies

8.1 Introduction

Already early in the history of Con \TeX T Taco Hoekwater wrote a module that dealt with bib \TeX databases in a Con \TeX T like way. Personally I never had to use a bibliography so I'm far from an expert in this area. However, going from some text database format to something typeset is generic enough for me to be involved.

The involvement started when MkIV showed up. Because quite some core mechanisms have been reimplemented, also some that the module used, a dedicated MkIV variant had to be made. This was not that hard to do as it mostly meant stripping code and replacing the specific reference mechanism by one using lists. That way we got a few bonus features but in general we can say that the module is downward compatible.

Already a while ago Taco and I discussed supporting bibliographies that use xml as format and although we have not settled on some standard it makes sense to explore the possibilities. The advantage of using xml is that we can use the built in subsystem for filtering and manipulating entries.

This chapter is dedicated to Thomas Schmitz who not only use bib \TeX but also has used MkIV xml right from the start and provides valuable feedback on both subsystems.

Keep in mind that eventually we will provide a high level interface so that users won't notice much of a difference unless they want to go beyond what they use now.

8.2 Sessions

As usual in Con \TeX T, we organize the featureset in such a way that we can group them and use several such sets in one documents without interference. It all starts by defining a session:

```
\definebibtexsession [somebibtex]
```

Next we register a couple of databases (from the beebe collection on \TeX Live:

```
\registerbibtexfile [somebibtex] [tugboat.bib]  
\registerbibtexfile [somebibtex] [komoedie.bib]
```

The files are loaded immediately and you can check this by looking at the log

where we get a report like:

```
mkiv lua stats : bibtex load time - 0.125 seconds (1524045 bytes, 2745 definitions, 7 shortcuts)
```

In a bibtex database there can be symbols (shortcuts to strings). Although these are expanded, it has no consequence for memory usage as Lua hashes its strings.

When we're done registering we need to prepare the session:¹¹

```
\preparebibtexsession [somebibtex] % [convert]
```

Say that we want to typeset a table with the publications of where we only list the the author and title. As we use the xml interface we do so using setups:

```
\startxmlsetups bibtex:one
  \starttabulate[|Bl|p|]
  \NC tag \NC \xmlatt{#1}{tag} \NC\NR
  \NC author\NC \xmlfilter{#1}{/field[@name='author']/context()} \NC\NR
  \NC title \NC \xmlfilter{#1}{/field[@name='title' ]/context()} \NC\NR
  \stoptabulate
\stopxmlsetups
```

We will now apply this setup to the loaded tree:

```
\startxmlsetups bibtex:bibtex
  \xmlfilter{#1}{
    /entry[@category='article']
    /field[@name='author' and (find(text(),'Hans Hagen')
      or find(text(),'Taco Hoekwater'))]
    ../command(bibtex:one)
  }
\stopxmlsetups
```

We now apply this setup to the session:

```
\applytobibtexsession[somebibtex][bibtex]
```

```
tag      hoekwater:tb19-3-256
author   Taco Hoekwater
title    Generating Type 1 fonts from MF sources
```

¹¹ The convert option will be discussed later.

tag hagen:tb25-1-108
author Hans Hagen
title The T_EX Live 2004 collection

tag hagen:tb19-3-304
author Hans Hagen
title The Calculator Demo, Integrating T_EX, MP, JavaScript and PDF

tag hagen:tb19-3-311
author Hans Hagen
title Visual Debugging in T_EX, Part 1: The Story

tag hagen:tb23-1-49
author Hans Hagen
title ConT_EXt, XML and T_EX: State of the art?

tag hagen:tb26-2-152
author Hans Hagen
title LuaT_EX: Howling to the moon

tag hoekwater:tb25-1-105
author Taco Hoekwater
title MetaPost developments

tag hagen:tb25-1-48
author Hans Hagen
title The state of ConT_EXt

tag hagen:tb22-3-136
author Hans Hagen
title Using T_EX for high end typesetting

tag hagen:tb22-3-118
author Hans Hagen
title Where will the odyssey bring us?

tag hagen:tb22-1-58
author Hans Hagen
title The status quo of the nts project

tag berdnikov:tb21-2-129
author Alexander Berdnikov and Hans Hagen and Taco Hoekwater and Bogusław Jackowski
title Even more MetaFun with MP: A request for permission

tag hagen:tb19-3-317
author Hans Hagen
title Visual Debugging in TeX, Part 2: The Macros

tag hagen:tb22-3-160
author Hans Hagen
title Using TeX to enhance your presentations

If this is the first time you see MkIV's `xml:n` action you might be confused by what happens here. When we apply the `bibtex` setup (the second argument), we expand a predefined setup that looks as follows:

```
\startxmlsetups bibtex
  \xmlregistereddokumentsetups{#1}{}
  \xmlsetsetup{#1}{bibtex|entry|field}{bibtex:*}
  \xmlmain{#1}
\stopxmlsetups
```

Here `#1` represents the root node of current database. Three elements are mapped to their own name, prefixed by `bibtex:`. In the previous examples we defined the `bibtex:bibtex` one, which will be applied to the root.

Here are a few more predefined setups:

```
\startxmlsetups bibtex:format
  \par
  \edef\currentbibxmlnode{#1}
  \xmlcommand{#1}{.}{bibtex:\currentbibtexformat:\xmlatt{#1}{category}}
  \par
\stopxmlsetups
```

```
\startxmlsetups bibtex:list
  \xmlfilter{#1}{/bibtex/entry/command(bibtex:format)}
\stopxmlsetups
```

```
\startxmlsetups bibtex:bibtex
  \xmlfilter{#1}{/entry/command(bibtex:format)}
\stopxmlsetups
```

The first one apply a setup to the current node (indicated by the period).

`bibtex:apa:article`

Such setups are defined elsewhere and you can imagine that they look more

complex than what we've seen so far. But you seldom have to deal with that.

The second and third setups apply the format to an entry. However, there is a subtle difference. The second one is called as follows:

```
\applytobibtexsession[somebibtex][bibtex:list]
```

As `bibtex:list` is a stand-alone setup, it will get the document root passed, and therefore we need to explicitly add that root, although the following two calls give the same results (watch the forward slashes):

```
\xmlfilter{#1}{/bibtex/entry/command(bibtex:format)}  
\xmlfilter{#1}{entry/command(bibtex:format)}
```

The `bibtex:bibtex` setup however, is using an indirect approach and only comes into action via the already mentioned `bibtex` setup. In that setup the `\xmlmain` command will expand the root element and when it sees the `bibtex` element, it will call the associated `bibtex:bibtex` setup. So here we need to call the `bibtex` setup.

```
\applytobibtexsession[somebibtex][bibtex]
```

Let's summarize what is needed to typeset a whole database:

```
\definebibtexsession [somebibtex]  
\registerbibtexfile [somebibtex] [tugboat.bib]  
\preparebibtexsession [somebibtex] [convert,strip]  
\applytobibtexsession [somebibtex] [bibtex:list]
```

Here we use the predefined `bibtex:list` filter. Of course you need to define commands that are used in the database.

8.3 The database

The xml database is quite simple and has the form (we omitted some fields):

```
<bibtex>  
  <entry tag="hagen:tb19-3-311" category="article">  
    <field name="number">3</field>  
    <field name="bibdate">Fri Jul 13 10:24:20 MDT 2007</field>  
    <field name="author">Hans Hagen</field>  
    <field name="journal">TUGboat</field>  
    <field name="title">{Visual Debugging in \TeX, Part 1: The Story}</field>  
    <field name="ISSN">0896-3207</field>
```

```

    <field name="year">1998</field>
    <field name="pages">311--317</field>
    <field name="volume">19</field>
  </entry>
</bibtex>

```

It is good to keep in mind that we lowercase the name and category attributes.

By default there are no setups for the one character elements but if you need then you have to use the bibtex namespace, e.g.:

```

\startxmlsetups bibtex:field
  \xmlflushcontext{#1}
\stopxmlsetups

```

8.4 Sorting

maybe also per session

We can sort entries. For that we need to define a sort setup. First we create a sort vector based on some fields. The first argument (bibtex) is the sort vector.

```

\startxmlsetups bibtex:entry:getkeys
  \xmladdsortentry{bibtex}{#1}
    {\xmlfilter{#1}{/field[@name='author']/text()}}
  \xmladdsortentry{bibtex}{#1}
    {\xmlfilter{#1}{/field[@name='year' ]/text()}}
  \xmladdsortentry{bibtex}{#1}
    {\xmllatt{#1}{tag}}
\stopxmlsetups

```

In the next setup we see this sorter being initialized. After that we filter some entries and add them to the to list of keys. Then we sort that list and flush it afterwards.

```

\startxmlsetups bibtex:entry:getkeys
  \xmladdsortentry{bibtex}{#1}
    {\xmlfilter{#1}{/field[@name='author']/text()}}
  \xmladdsortentry{bibtex}{#1}
    {\xmlfilter{#1}{/field[@name='year' ]/text()}}
  \xmladdsortentry{bibtex}{#1}
    {\xmllatt{#1}{tag}}
\stopxmlsetups

```

The flusher simply shows some fields. You can do anything you want here with the content.

```
\startxmlsetups bibtex:entry:flush
  \xmlfilter{#1}{/field[@name='author']/context()} / %
  \xmlfilter{#1}{/field[@name='year' ]/context()} / %
  \xmlatt{#1}{tag}\par
\stopxmlsetups
```

The setup that brings this all together is applied to the whole tree with the following command.

```
\xmlsetup{bibtex:somebibtex}{xml:bibtex:sorter}
```

The result is:

```
Don Knuth / 1984 / knuth:tb5-1-67
Donald E. Knuth / 1984 / knuth:tb5-1-4
Donald E. Knuth / 1984 / knuth:tb5-2-105
Donald E. Knuth / 1985 / knuth:tb6-1-36
Donald E. Knuth / 1986 / knuth:tb7-2-101
Donald E. Knuth / 1987 / knuth:tb8-2-135
Donald E. Knuth / 1987 / knuth:tb8-3-309
Donald E. Knuth / 1988 / knuth:tb9-2-152
Donald E. Knuth / 1989 / knuth:tb10-3-325
Donald E. Knuth / 1989 / knuth:tb10-4-529
Donald E. Knuth / 1990 / knuth:tb11-4-489
Donald E. Knuth / 1993 / knuth:tb14-4-387
Donald E. Knuth / 1996 / knuth:tb17-1-29
Donald Knuth and Pierre MacKay / 1987 / knuth:tb8-1-14
Donald Knuth / 1981 / knuth:tb2-3-5
Donald Knuth / 1982 / knuth:tb3-1-10
Donald Knuth / 1983 / knuth:tb4-2-64
Donald Knuth / 1986 / knuth:tb7-2-95
Donald Knuth / 1987 / knuth:tb8-1-6
Donald Knuth / 1987 / knuth:tb8-1-73
Donald Knuth / 1987 / knuth:tb8-2-210
Donald Knuth / 1987 / knuth:tb8-2-217
Donald Knuth / 1989 / knuth:tb10-1-8
Donald Knuth / 1989 / knuth:tb10-1-31
Donald Knuth / 1990 / knuth:tb11-1-13
Donald Knuth / 1990 / knuth:tb11-2-165
Donald Knuth / 1990 / knuth:tb11-4-497
Donald Knuth / 1990 / knuth:tb11-4-499
```

Donald Knuth / 1991 / knuth:tb12-2-313

You can call up the list of keys with

```
\xmlshowsorter{bibtex}
```

In our case this gives:

n	id	entry 1	entry 2	entry 3
1	324	Donald Knuth	1991	knuth:tb12-2-313
2	1397	Donald Knuth	1983	knuth:tb4-2-64
3	4173	Donald Knuth	1981	knuth:tb2-3-5
4	5247	Donald Knuth	1987	knuth:tb8-1-6
5	5943	Donald Knuth and Pierre MacKay	1987	knuth:tb8-1-14
6	7773	Donald Knuth	1989	knuth:tb10-1-8
7	10665	Donald Knuth	1990	knuth:tb11-1-13
8	10871	Donald Knuth	1986	knuth:tb7-2-95
9	11248	Donald E. Knuth	1990	knuth:tb11-4-489
10	12236	Donald Knuth	1990	knuth:tb11-4-497
11	12535	Donald E. Knuth	1984	knuth:tb5-2-105
12	12665	Donald E. Knuth	1993	knuth:tb14-4-387
13	12925	Donald E. Knuth	1988	knuth:tb9-2-152
14	14902	Donald E. Knuth	1987	knuth:tb8-2-135
15	15161	Donald Knuth	1987	knuth:tb8-2-217
16	21952	Donald Knuth	1990	knuth:tb11-4-499
17	23934	Donald Knuth	1987	knuth:tb8-2-210
18	24128	Donald Knuth	1987	knuth:tb8-1-73
19	26859	Donald E. Knuth	1989	knuth:tb10-4-529
20	28026	Donald Knuth	1989	knuth:tb10-1-31
21	28091	Donald E. Knuth	1996	knuth:tb17-1-29
22	28572	Donald E. Knuth	1986	knuth:tb7-2-101
23	28624	Donald E. Knuth	1984	knuth:tb5-1-4
24	31473	Donald E. Knuth	1985	knuth:tb6-1-36
25	34586	Donald E. Knuth	1987	knuth:tb8-3-309
26	34911	Don Knuth	1984	knuth:tb5-1-67
27	34950	Donald Knuth	1990	knuth:tb11-2-165
28	34976	Donald Knuth	1982	knuth:tb3-1-10
29	35196	Donald E. Knuth	1989	knuth:tb10-3-325

8.5 Encodings

It is a sure bet that many existing databases will use the traditional \TeX accent

building commands. As in MkIV we live in an Unicode universe, such commands are translated into utf sequences when the database is loaded. When we pass the convert options to the preparation command, the entries will be cleaned up and accent commands will be replaced by proper utf sequences. This helps the sorter.

8.6 Messed up entries

As the bib \TeX fields contains \TeX code we need to process the content as \TeX . This is why in the previous examples we applied the context() finalizer. The fact that we have \TeX code means that such databases are rather bound to some macro package. For our purpose we had to define a few macros:

```
\startsetups bibtex-commands
  \def\MF {MF}
  \def\MP {MP}
  \def\TUB {TUGboat}
  \def\Mc {Mac}
  \def\slett{\tt}
  \let\acro\firstofoneargument
\stopsetups
```

You can best do this grouped is that there is no interference with existing code. You can collect definitions in a setup or buffer and flush that one inside the group.

However, we also provide another method. A second argument to the preparation command gives options. Examples of options are convert, which converts entries to proper utf, and strip which converts commands and strips redundant braces.

```
\preparebibtexsession [somebibtex] [convert,strip]
```

All commands are mapped onto \bibtexcommand which defaults to using predefined local commands. You predefine such a local command with:

```
\defbibtexcommand\MF {MF}
\defbibtexcommand\MP {MP}
\defbibtexcommand\TUB {TUGboat}
\defbibtexcommand\Mc {Mac}
\defbibtexcommand\slett {\tt}
\defbibtexcommand\acro#1{#1}
```

If you use a database like tugboat.bib you will need quite some more defini-

tions. Unknown commands are reported on the console. When a command is available in ConT_EXt it will be used unless a specific one is defined.

Here's a setup that shows what goes on inside:

```
\startxmlsetups bibtex:show
  \xmlshow{#1}
\stopxmlsetups

\applytobibtexsession[somebibtex][bibtex:show]
```

8.7 Traditional usage

In this section we will describe how you can use this approach as a drop in for the traditional, pure T_EX based one.

9 Building paragraphs

9.1 Introduction

You enter the den of the Lion when you start messing around with the parbuilder. Actually, as \TeX does a pretty good job on breaking paragraphs into lines I never really looked in the code that does it all. However, the Oriental \TeX project kind of forced it upon me. In the chapter about font goodies an optimizer is described that works per line. This method is somewhat similar to expansion level one support in the sense that it acts independent of the parbuilder: the split off (best) lines are postprocessed. Where expansion involves horizontal scaling, the goodies approach does with (Arabic) words what the original HZ approach does with glyphs.

It would be quite some challenge (at least for me) to come up with solutions that looks at the whole paragraph and as the per-line approach works quite well, there is no real need for an alternative. However, in September 2008, when we were exploring solutions for Arabic par building, Taco converted the parbuilder into Lua code and stripped away all code related to hyphenation, protrusion, expansion, last line fitting, and some more. As we had enough on our plate at that time, we never came to really testing it. There was even less reason to explore this route because in the Oriental \TeX project we decided to follow the “use advanced OpenType features” route which in turn lead to the ‘replace words in lines by narrower or wider variants’ approach.

However, as the code was laying around and as we want to explore further I decided to pick up the parbuilder thread. In this chapter some experiences will be discussed. The following story is as much Taco's as mine.

9.2 Cleaning up

In retrospect, we should not have been too surprised that the first approximation was broken in many places, and for good reason. The first version of the code was a conversion of the C code that in turn was a conversion from the original interwoven Pascal code. That first conversion still looked quite C-ish and carried interesting bit and pieces of C-macros, C-like pointer tests, interesting magic constants and more.

When I took the code and Lua-fied it nearly every line was changed and it took Taco and me a bit of reverse engineering to sort out all problems (thank you Skype). Why was it not an easy task? There are good reasons for this.

- The parbuilder (and related hpacking) code is derived from traditional \TeX and has bits of pdf \TeX , Aleph (Omega), and of course Lua \TeX .

- The advocated approach to extending \TeX has been to use change files which means that a coder does not see the whole picture.
- Originally the code is programmed in the literate way which means that the resulting functions are build stepwise. However, the final functions can (and have) become quite large. Because Lua \TeX uses the woven (merged) code indeed we have large functions. Of course this relates to the fact that successive \TeX engines have added functionality. Eventually the source will be webbed again, but in a more sequential way.
- This is normally no big deal, but the Aleph (Omega) code has added a level of complexity due to directional processing and additional begin and end related boxes.
- Also the ε - \TeX extension that deals with last line fitting is interwoven and uses goto's for the control flow. Fortunately the extensions are driven by parameters which makes the related code sections easy to recognize.
- The pdf \TeX protrusion extension adds code to glyph handling and discretionary handling. The expansion feature does that too and in addition also messes around with kerns. Extra parameters are introduced (and adapted) that influence the decisions for breaking lines. There is also code originating in pdf \TeX which deals with poor mans grid snapping although that is quite isolated and not interwoven.
- Because it uses a slightly different way to deal with hyphenation, Lua \TeX itself also adds some code.
- Tracing is sort of interwoven in the code. As it uses goto's to share code instead of functions, one needs to keep a good eye on what gets skipped or not.

I'm pretty sure that the code that we started with looks quite different from the original \TeX code if it had been translated into C. Actually in modern \TeX compiling involves a translation into C first but the intermediate form is not meant for human eyes. As the Lua \TeX project started from that merged code, Taco and Hartmut already spend quite some time on making it more readable. Of course the original comments are still there.

Cleaning up such code takes a while. Because both languages are similar but also quite different it took some time to get compatible output. Because the C code uses macros, careful checking was needed. Of course Lua's table model and local variables brought some work as well. And still the code looks a bit C-ish. We could not divert too much from the original model simply because

it's well documented.

When moving around code redundant tests and orphan code has been removed. Future versions (or variants) might as well look much different as I want more hooks, clearly split stages, and convert some linked list based mechanism to Lua tables. On the other hand, as already much code has been written for ConTeXt MkIV, making it all reasonable fast was no big deal.

9.3 Expansion

The original C-code related to protrusion and expansion is not that efficient as many (redundant) function calls take place in the linebreaker and packer. As most work related to fonts is done in the backend, we can simply stick to width calculations here. Also, it is no problem at all that we use floating point calculations (as Lua has only floats). The final result will look okay as the original hpack routine will nicely compensate for rounding errors as it will normally distribute the content well enough. We are currently compatible with the regular par builder and protrusion code, but expansion gives different results (actually not worse).

The Lua hpacker follows a different approach. And let's admit it: most TeXies won't see the difference anyway. As long as we're cross platform compatible it's fine.

It is a well known fact that character expansion slows down the parbuilder. There are good reasons for this in the pdfTeX approach. Each glyph and intercharacter kern is checked a few times for stretch or shrink using a function call. Also each font reference is checked. This is a side effect of the way pdfTeX backend works as there each variant has its own font. However, in LuaTeX, we scale inline and therefore don't really need the fonts. Even better, we can get rid of all that testing and only need to pass the eventual `expansion_ratio` so that the backend can do the right scaling. We will prototype this in the Lua version¹² and we feel confident about this approach it will be backported into the C code base. So eventually the C might become a bit more readable and efficient.

Intercharacter kerning is dealt with somewhat strange. When a kern of subtype zero is seen, and when it's neighbours are glyphs from the same font, the kern gets replaced by a scaled one looked up in the font's kerning table. In the parbuilder no real replacement takes place but as each line ends up in the hpack routine (where all work is simply duplicated and done again) it really

¹² For this Hartmuts has adapted the backend code has to honour this field in the glyph and kern nodes.

gets replaced there. When discussing the current approach we decided that manipulating intercharacter kerns while leaving regular spacing untouched is not really a good idea so there will be an extra level of configuration added to Lua \TeX :¹³

- 0 no character and kern expansion
- 1 character and kern expansion applied to complete lines
- 2 character and kern expansion as part of the par builder
- 3 only character expansion as part of the par builder (new)

You might wonder what happens when you unbox such a list: the original font references have been replaced as are the kerns. However, when repackaged again, the kerns are replaced again. In traditional \TeX , indeed rekerneling might happen when a paragraph is repackaged (as different hyphenation points might be chosen and ligature rebuilding etc. has taken place) but in Lua \TeX we have clearly separated stages. An interesting side effect of the conversion is if that we really have to wonder what certain code does and if it's still needed.

9.4 Performance

We had already noticed that the Lua variant was not that slow so after the first cleanup it was time to do some tests. We used our regular `tuftex.tex` test file. This happens to be a worst case example because each broken line ends with a comma or hyphen and these will hang into the margin when protruding is enabled. So the solution space is rather large (an example will be shown later).

Here are some timings of the March 26, 2010 version. The test is typeset in a box so no shipout takes place. We're talking of 1000 typeset paragraphs. The times are in seconds and between parentheses the speed relative to the regular parbuilder is mentioned.

	native	lua	lua + hpack
normal	1.6	8.4 (5.3)	9.8 (6.1)
protruding	1.7	14.2 (8.4)	15.6 (9.2)
expansion	2.3	11.4 (5.0)	13.3 (5.8)
both	2.9	19.1 (6.6)	21.5 (7.4)

For a regular paragraph the Lua variant (currently) is 5 times slower and about 6 times when we use the Lua hpacker, which is not that bad given that it's interpreted code and that each access to a field in a node involves a function call. Actually, we can make a dedicated hpacker as soem code can be omitted,

¹³ As I more and more run into books typeset (not by \TeX) with a combination of character expansion and additional intercharacter kerning I've been seriously thinking of removing support for expansion from Con \TeX t MkIV. Not all is progress especially if it can be abused.

The reason why the protruding is relative slow is that we have quite some protruding characters in the test text (many commas and potential hyphens) and therefore we have quite some lookups and calculations. In the C variant much of that is inlined by macros.

Will things get faster? I'm sure that I can boost the protrusion code and probably the rest as well but it will always be slower than the built in function. This is no problem as we will only use the Lua variant for experiments and special purposes. For that reason more MkIV like tracing will be added (some is already present) and more hooks will be provided once that the builder is more compartmentalized. Also, future versions of LuaTeX will pass around paragraph related parameters differently so that will have impact on the code as well.

9.5 Usage

The basic parbuilder is enabled and disabled as follows:¹⁴

```
\definefontfeature[example][default][protrusion=pure]
\definedfont[Serif*example]
\setupalign[hanging]

\startparbuilder[basic]
  \startcolor[blue]
  \input tufte
  \stopcolor
\stopparbuilder
```

This results in:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsise, winnow the wheat from the chaff and separate the sheep from the goats.

There are a few tracing options in the parbuilders namespace but these are not stable yet.

¹⁴ I'm not sure yet if the parbuilder has to do automatic grouping.

9.6 Conclusion

The module started working quiet well around the time that Peter Gabriels “Scratch My Back” ended up in my Squeezecenter: modern classical interpretations of some of his favourite songs. I must admit that I scratched the back of my head a couple of times when looking at the code below. It made me realize that a new implementation of a known problem indeed can come out quite different but at the same time has much in common. As with music it's a matter of taste which variant a user likes most.

At the time of this writing there is still work to do. For instance the large functions need to be broken into smaller steps. And of course more testing is needed.

10 Tagged PDF

10.1 Introduction

Occasionally users asked me if ConTeXt can produce tagged pdf and the answer to that has been: I'll implement it when I need it. However, users tell me that publishers more and more demand tagged pdf files, although one might wonder what for, maybe except for accessibility. Another reason for not having spent too much time on it before is that the specification was not that inviting.

Anyhow, when I saw Ross Moore¹⁵ presenting tagged math at TUG 2010, I decided to look up the spec once more and see if I could get into the mood to implement tagging. Before I started it was already clear that there were a couple of boundary conditions:

- Tagging should not put a burden on the user but users should be able to tag themselves.
- Tagging should not slow down a run too much; this is no big deal as one can postpone tagging till the last run.
- Tagging should in no way interfere with typesetting, so no funny nodes should be injected.
- Tagging should not make the code look worse, neither the document source, not the low level ConTeXt code.

And of course implementing it should not take more than a few days work, certainly not in an exceptional hot summer.

You can 'google' for one of Ross's documents (like `DML_002-2009-1_12.pdf`) to see how a document source looks at his end using a special version of pdfTeX. However, the version on my machine didn't support the shown primitives, so I could not see what was happening under the hood. Unfortunately it is quite hard to find a proper tagged document so we have only the reference manual as starting point. As the pdfTeX approach didn't look that pleasing anyway, I just started from scratch.

Tags can help Acrobat Reader when reading out the text loud. But you cannot browse the structure in this free program and as not all users have the professional version of Acrobat, the fact that a document has structure can go unnoticed. Add to that the fact that the overhead in terms of bytes is quite large as many more objects are generated, and you will understand why this feature is not enabled by default.

¹⁵ He is often exploring the boundaries of pdf, Unicode and evolving techniques related to math publishing so you'd best not miss his presentations when you are around.

10.2 Implementation

So, what does tagging boil down to? We can best look at how tagged information is shown in Acrobat. Figure 10.1 shows the content tree that has been added (automatically) to a document while figure 10.2 shows a different view.

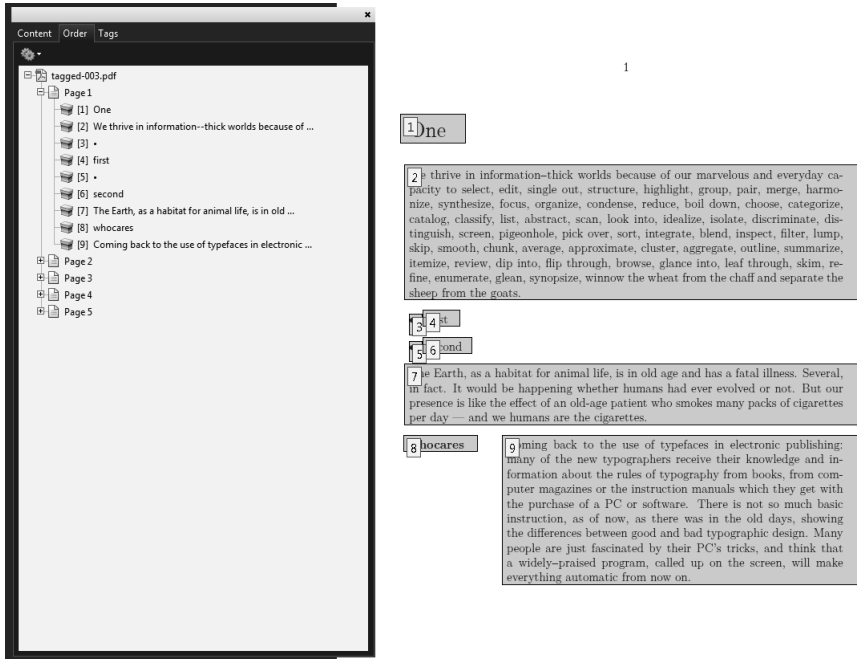


Figure 10.2 Acrobat showing the tag order.

In order to get that far, we have to do the following:

- Carry information with (typeset) text.
- Analyse this information when shipping out pages.
- Add a structure tree to the page.
- Add relevant information to the document.

That first activity is rather independent of the other three and we can use that information for other purposes as well, like identifying where we are in the document. We carry the information around using attributes. The last three activities took a bit of experimenting mostly using the “Example of Logical Structure” from the pdf standard 32000-1:2008.

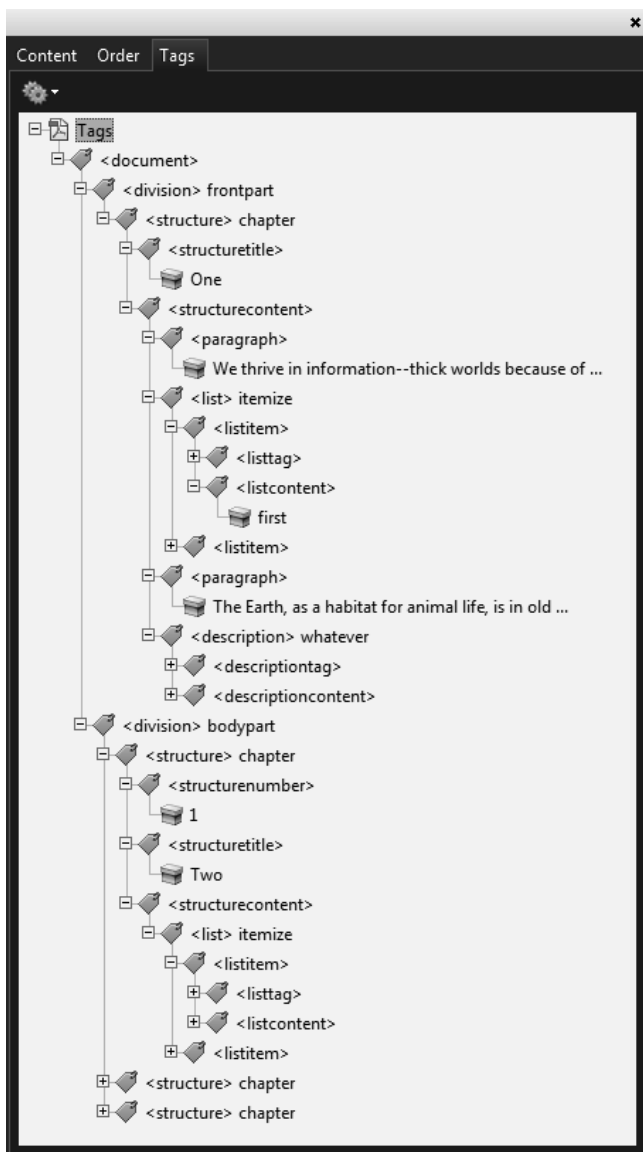


Figure 10.1 A tag list in Acrobat.

This resulted in tagging framework that uses explicit tags. In that the user is responsible for the tagging:

```
\setupstructure[state=start,method=none]

\starttext

\startelement[document]

    \startelement[chapter]
        \startelement[p] \input davis \stopelement \par
    \stopelement

    \startelement[chapter]
        \startelement[p] \input zapf \stopelement \par
        \startelement[whatever]
            \startelement[p] \input tufte \stopelement \par
            \startelement[p] \input knuth \stopelement \par
        \stopelement
    \stopelement

    \startelement[chapter]
        oeps
        \startelement[p] \input ward \stopelement \par
    \stopelement

\stopelement

\stoptext
```

However, this is not much fun so we also provide an automated variant. In the previous example we explicitly turned of automated tagging by setting method to none. By default it has the value auto.

```
\setupstructure[state=start] % method=auto is default

\definedescription[whatever]

\starttext

\startfrontmatter
    \startchapter[title=One]
        \startparagraph \input tufte \stopparagraph
        \startitemize
```

```

        \startitem first \stopitem
        \startitem second \stopitem
    \stopitemize
    \startparagraph \input ward \stopparagraph
    \startwhatever {Herman Zapf} \input zapf \stopwhatever
\stopchapter

```

```
\stopfrontmatter
```

```
\startbodymatter
```

```
.....
```

If you use commands like `\chapter` you will not get the desired results. Of course these can be supported but there is no real reason for it, as in MkIV we advise to use the start-stop variant.

It will be clear that this kind of automated tagging brings with it a couple of extra commands deep down in ConT_EXt and there (of course) we use symbolic names for tags, so that one can overload the built in mapping.

```
\setuptaglabeltext[en][document=text]
```

As with other features inspired by viewer functionality, the implementation of tagging is independent of the backend. For instance, we can tag a document and access tagging information at the T_EX end. The backend drivers code maps tags to relevant pdf constructs. First of all, we just map the tags used at the ConT_EXt end onto themselves. But, as validators expect certain names, we use the pdf rolemap feature to map them to (less interesting) names. The next list shows the currently used internal names with the pdf ones between parentheses.

construct (Span) delimited (Quote) delimitedblock (BlockQuote) description (Div) descriptioncontent (Div) descriptionsymbol (Span) descriptiontag (Div) division (Div) document (Div) float (Div) floatcaption (Caption) floatcontent (P) floattag (Span) floattext (Span) formula (Div) formulacontent (P) formulacontent (Div) formulaset (Div) formulatag (Span) image (P) item (Li) itemcontent (LBody) itemgroup (L) itemtag (Lbl) link (Link) list (TOC) listcontent (P) listdata (P) listitem (TOCI) listpage (Reference) listtag (Lbl) margintext (Span) margintextblock (Span) math (Div) merror (Span) mfrac (Span) mi (Span) mn (Span) mo (Span) mover (Span) mgraphic (P) mroot (Span) mrow (Span) ms (Span) msqrt (Span) msub (Span) msubsup (Span) msup (Span) mtext (Span) munder (Span) munderover (Span) paragraph (P) register (Div) registerentries (Div) registerentry (Span) registerpage (Span) registerpages (Span) registersection (Div) registersee (Span) regis-

tertag (Span) section (Sect) sectioncontent (Div) sectionnumber (H) sectiontitle (H) subformula (Div) subsentence (Span) table (Table) tablecell (TD) tablerow (TR) tabulate (Table) tabulatecell (TD) tabulaterow (TR) verbatim (Code) verbatimblock (Code) verbatimline (Code)

So, the internal ones show up in the tag trees as shown in the examples but applications might use the rolemap which normally has less detail.

Because we keep track of where we are, we can also use that information for making decisions.

```
\doifinlementelse{structure:section}          {yes} {no}
\doifinlementelse{structure:chapter}         {yes} {no}
\doifinlementelse{division:*-structure:chapter} {yes} {no}
\doifinlementelse{division:*-structure:*}     {yes} {no}
```

You can use the `*` as wildcard. The elements are separated by a `-`. If you don't know what tags are used, you can always enable the tag related tracker:

```
\enabletrackers[structure.tags]
```

This tracker reports the identified element chains to the console and log.

10.3 Special care

Of course there are a few complications. First of all the tagging model sort of contradicts the concept of a nicely typeset document where structure and outcome are not always related. Most $\text{T}_{\text{E}}\text{X}$ users are aware of the fact that $\text{T}_{\text{E}}\text{X}$ does not have spaces and does a great job on kerning and hyphenation. The tagging machinery on the other hand uses a rather dumb model of strings separated by spaces.¹⁶ But anyhow we could trick $\text{T}_{\text{E}}\text{X}$ into providing the right information to the backend so that words get nicely separated. The non-optimized function that does this looks as follows:

```
function injectspaces(head)
  local p
  for n in node.traverse(head) do
    local id = n.id
    if id == node.id("glue") then
      if p and p.id == node.id("glyph") then
        local g = node.copy(p)
        local s = node.copy(n.spec)
```

¹⁶ The search engine on the other hand is rather clever on recognizing words.

```

        g.char, n.spec = 32, s
        p.next, g.prev = g, p
        g.next, n.prev = n, g
        s.width = s.width - g.width
    end
    elseif id == node.id("hlist") or id == node.id("vlist") then
        injectspaces(n.list,attribute)
    end
    p = n
end
end

```

Here we squeeze in a space (given that it is in the font which it normally is when you use Con \TeX t) and compensate the glue. Given that your page sits in box 255, you can do this just before shipping the page out:

```
injectspaces(tex.box[255].list)
```

Then there are the so called suspects: things on the page that are not related to structure at all. One is supposed to tag these specially so that the built-in reading equipment is not confused. So far we could get around them simply because they don't get tagged at all and therefore are not seen anyway. This might as well be enough of a precaution.

Of course we need to deal with mathematics. Fortunately the presentation MathML model is rather close to \TeX and so we can map onto that. After all we don't need to care too much about back-mapping here. The currently present code is rather experimental and might get extended or thrown out in favour of inline mathml. Figure 10.3 demonstrates that a first approach does not even look that bad. In future versions we might deal with table like math constructs, like matrices.

This is a typical case where more energy has to be spent on driving the voice of Acrobat but I will do that when we find a good reason.

As mentioned, it will take a while before all relevant constructs in Con \TeX t support tagging, but support is already quite complete. Some screen dumps are included as example at the end.

10.4 Conclusion

Surprisingly implementing all this didn't take that much work. Of course detailed automated structure support from the complete Con \TeX t kernel will take

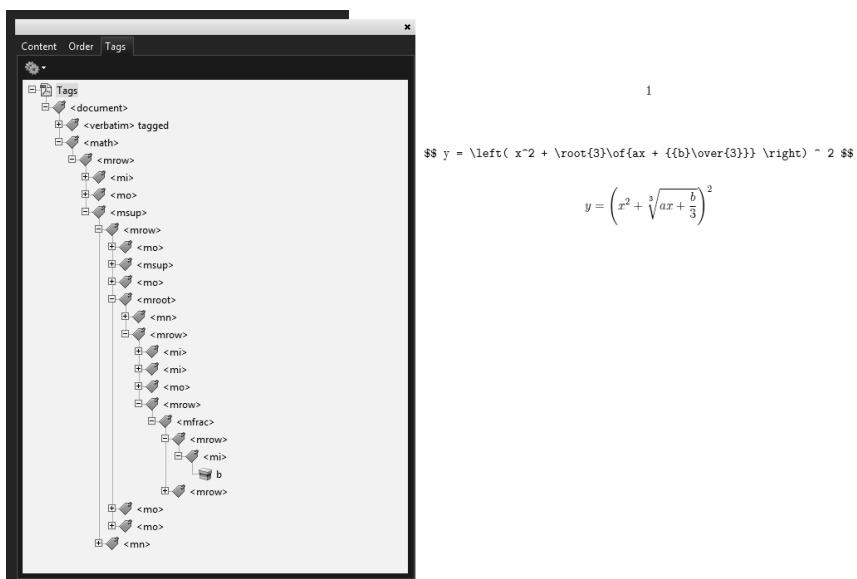


Figure 10.3 Experimental math tagging.

some time to get completed, but that will be done on demand and when we run into missing bits and pieces. It's still not decided to what extent alternate representations and alternate texts will be supported. Experiments with the reading loud machinery are not satisfying yet but maybe it just can't get any better. It would be nice if we could get some tags being announced without overloading the content, that is: without using ugly hacks.

And of course, code like this is never really finished if only because pdf evolves. Also, it is yet another nice test case and torture test for Lua \TeX and it helps us to surface buglets and oversights.

10.5 Some more examples

In Con \TeX t we have user definable verbatim environments. As with other user definable environments we show the specific instance as comment next to the structure component. See figure 10.4. Some examples of tables are shown in figure 10.5. Future versions will have a bit more structure. Tables of contents (see figure 10.6) and registers (see figure 10.7) are also tagged. One might wonder what the use is of this. In Figure 10.8 we see some examples of floats. External images as well as METAPOST graphics are tagged as such. This example also shows an example of a user environment, in this case:

`\definestartstop[notabene][style=\bf]`

In a similar fashion footnotes end up in the structure tree, but in the typeset document they move around (normally forward when there is no room).

1 chapter

test oeps test **whow** test

test

`\whatever[goes]{here}`

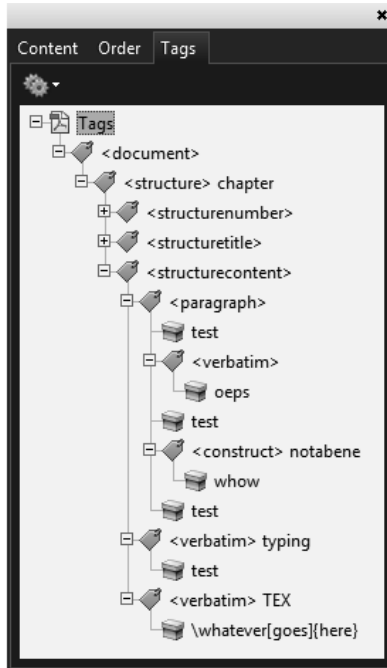
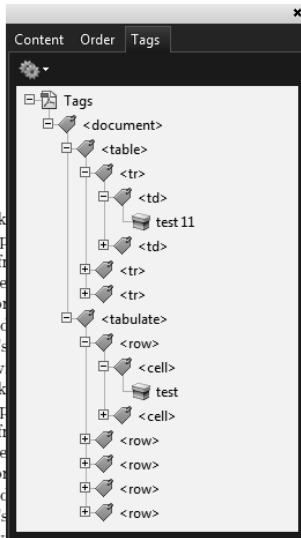


Figure 10.4 Verbatim, including dedicated instances.

test 11	test 12
test 21	test 22
test 33	

test Coming back to the new typography from the typography file which they get basic instructions between good by their PC's the screen, with

test Coming back to the new typography from the typography file which they get basic instructions between good by their PC's the screen, with



publishing: many of the information about the rules of the instruction manuals. There is not so much bashowing the differences people are just fascinated by a program, called up on on.

publishing: many of the information about the rules of the instruction manuals. There is not so much bashowing the differences people are just fascinated by a program, called up on on.

Figure 10.5 Natural tables as well as the tabulate mechanism is supported.

Contents

1	One	2
1.1	alpha	2
1.2	beta	2
1.3	gamma	2
1.4	delta	2

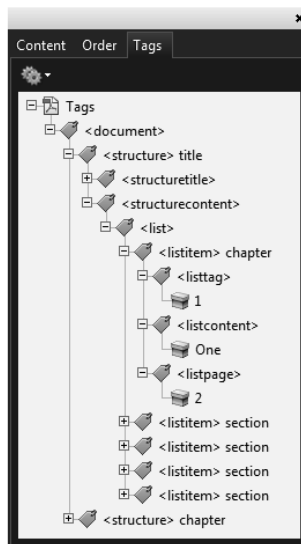


Figure 10.6 Tables of content with specific entries tagged.

Index

o one 1, 2
t two 1, 2

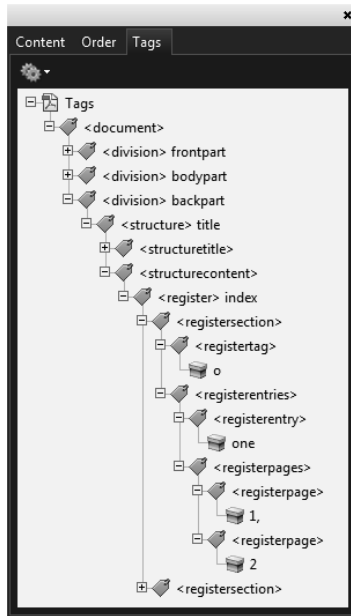


Figure 10.7 A detailed view of registered is provided.

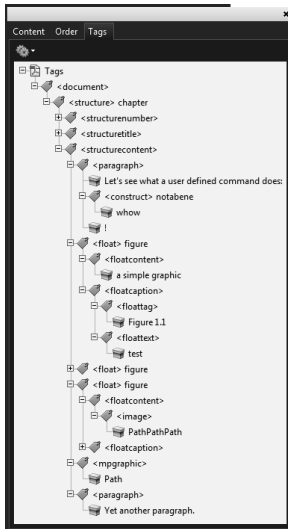
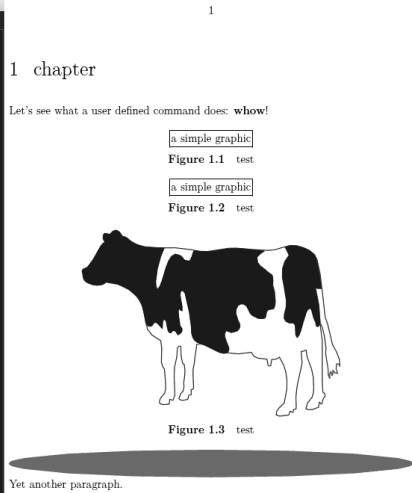


Figure 10.8 Floats tags end up in text stream. Watch the user defined construct.



sheep from the

- first¹
- second

The Earth, as a
in fact. It would
presence is like
per day — and

who cares

¹ test

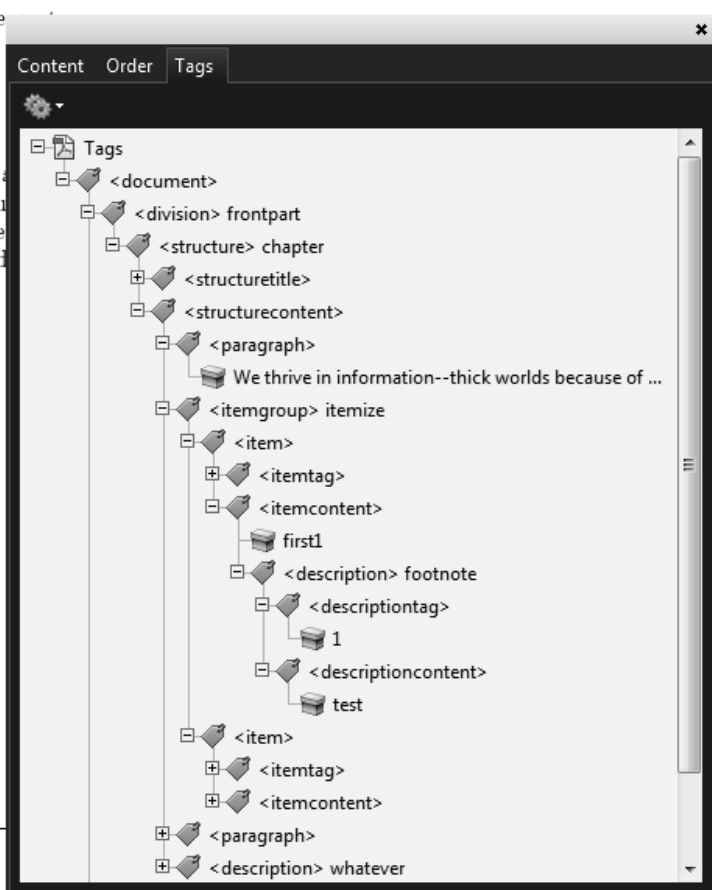


Figure 10.9 Footnotes are shown at the place in the input (flow).

11 Including pages

11.1 Introduction

It is tempting to add more and more features to the backend code of the engine but it is not really needed. Of course there are features that can best be supported natively, like including images. In order to include pdf images in LuaTeX the backend uses a library (xpdf or poppler) that can load an page from a file and embed that page into the final pdf, including all relevant (indirect) objects needed for rendering. In LuaTeX an experimental interface to this library is included, tagged as epdf. In this chapter I will spend a few words on my first attempt to use this new library.

11.2 The library

The interface is rather low level. I got the following example from Hartmut (who is responsible for the LuaTeX backend code and this library).

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc:getCatalog()
local pag = cat:getPage(3)
local box = pag:getMediaBox()

local w = pag:getMediaWidth()
local h = pag:getMediaHeight()
local n = cat:getNumPages()
local m = cat:readMetadata()

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ", box.x1, box.x2, box.y1, box.y2)
```

As you see, there are accessors for each interesting property of the file. Of course such an interface needs to be extended when the pdf standard evolves. However, once we have access to the so called catalog, we can use regular accessors to the dictionaries, arrays and other data structures. So, in fact we don't need a full interface and can draw the line somewhere.

There are a couple of things that you normally does not want to deal with. A pdf file is in fact just a collections of objects that form a tree and each object can be reached by an index using a table that links the index to a position in the file. You don't want to be bothered with that kind of housekeeping indeed. Some data in the file, like page objects and annotations are organized in a

tree form that one does not want to access in that form, so again we have something that benefits from an interface. But the majority of the objects are simple dictionaries and arrays. Streams (these hold the document content, image data, etc.) are normally not of much interest, but the library provides an interface as you can bet on needing it someday. The library also provides ways to extend the loaded pdf file. I will not discuss that here.

Because in ConTeXt we already have the `lpdf` library for creating pdf structures, it makes sense to define a similar interface for accessing pdf. For that I wrote a wrapper that will be extended in due time (read: depending on needs). The previous code now looks as follows:

```
local doc = epdf.open("luatexref-t.pdf")
local cat = doc.Catalog
local pag = cat.Pages[3]
local box = pag.MediaBox

local llx, lly, urx, ury = box[1], box[2] box[3], box[4]

local w = urx - llx -- or: box.width
local h = ury - lly -- or: box.height
local n = cat.Pages.size
local m = cat.Metadata.stream

print("nofpages: ", n)
print("metadata: ", m)
print("pagesize: ", w .. " * " .. h)
print("mediabox: ", llx, lly, urx, ury)
```

If we write code this way we are less dependent on the exact api, especially because the `epdf` library uses methods to access the data and we cannot easily overload method names in there. When you look at the `box`, you will see that the natural way to access entries is using a number. As a bonus we also provide the width and height entries.

11.3 Merging links

It has always been on my agenda to add the possibility to carry the (link) annotations with an included page from a document. This is not that much needed in regular document, but it can be handy when you use ConTeXt to assemble documents. In any case, such a merge has to happen in such a way that it does not interfere with other links in the parent document. Supporting this in the engine is no option as each macro package follows its own approach to referencing and interactivity. Also, demands might differ and one would end

up with a lot of (error prone) configurability. Of course we want scaled pages to behave well too.

Implementing the merge took about a day and most of that time was spent on experimenting with the `epdf` library and making the first version of the wrapper. I definitely had expected to waste more time on it. So, this is yet another example of extensions that are quite doable in the Lua- \TeX mix. Of course it helps that the Con \TeX t graphic inclusion code provides enough information to integrate such a feature. The merge is controlled by the interaction key, as shown here:

```
\externalfigure[somefile.pdf][page=1,scale=700,interaction=yes]
\externalfigure[somefile.pdf][page=2,scale=600,interaction=yes]
\externalfigure[somefile.pdf][page=3,scale=500,interaction=yes]
```

You can finetune the merge by providing a list of options to the interaction key but that's still somewhat experimental. As a start the following links are supported.

- internal references by name (often structure related)
- internal references by page (like on tables of contents)
- external references by file (optionally by name and page)
- references to uri's (normally used for webpages)

When users like this functionality (or when I really need it myself) more types of annotations can be added although support for JavaScript and widgets doesn't make much sense. On the other hand, support for destinations is currently somewhat simplified but at some point we will support the relevant zoom options.

The implementation is not that complex:

- check if the included page has annotations
- loop over the list of annotations and determine if an annotation is supported (currently links)
- analyze the annotation and overlay a button using the destination that belongs to the annotation

Now, the reason why we can keep the implementation so simple is that we just map onto existing Con \TeX t functionality. And, as we have a rather integrated support for interactive actions, only a few basic commands are involved. Although we could do that all in Lua, we delegate this to \TeX . We create a layer that we put on top of the image. Links are put onto this layer using the equivalent of:

```

\setlayer
  [epdflinks]
  [x=...,y=...,preset=leftbottom]
  {\button
    [width=...,height=...,offset=overlay,frame=off]
    {}% no content
    [...]}

```

The `\button` command is one of those interaction related commands that accepts any action related directive. In this first implementation we see the following destinations show up:

```

somelocation
url(http://www.pragma-ade.com)
file(somefile)
somefile:somelocation
somefile:page(10)

```

References to pages become named destinations and are later resolved to page destinations again, depending on the configuration of the main document. The links within an included file get their own namespace so (hopefully) they will not clash with other links.

We could use lower level code which is faster but we're not talking of time critical code here. At some point I might optimize the code a bit but for the moment this variant gives us some tracing options for free. Now, the nice thing about using this approach is that the already existing cross referencing mechanisms deal with the details. Each included page gets a unique reference so references to not included pages are ignored simply because they cannot be resolved. We can even consider overloading certain types of links or ignoring named destinations that match a specific pattern. Nothing is hard coded in the engine so we have complete freedom of doing that.

11.4 Merging layers

When including graphics from other applications it might be that they have their content organized in layers (that then can be turned on or off). So it will be no surprise that on the agenda is merging layer information: first a straightforward inclusion of optional content dictionaries, but it might make sense to parse the content stream and replace references to layers by those that are relevant in the main document. Especially when graphics come from different sources and layer names are inconsistent some manipulation might be needed so maybe we need more detailed control. Implementing this is no big deal and mostly a matter of figuring out a clean and simple user interface.

12 Exporting XML

12.1 Introduction

Every now and then on the the mailing list users ask if ConT_EXt can produce html instead of for instance pdf, and the answer has always been unsatisfying. In this chapter I will present the MkIV way of doing this.

12.2 The clumsy way

My favourite answer to the question about how to produce html (or more general xml as it can be transformed) has always been: “I’d just typeset it!”. Take:

```
\def\MyChapterCommand#1#2{<h1>#2</h1>}  
\setuphead[chapter][command=\MyChapterCommand]
```

Here `\chapter{Hello World}` will produce:

```
<h1>Hello World</h1>
```

Now imagine that you hook such commands into all relevant environment and that you use a style with no header and footer lines. You use a large page (A2) and a small monospaced font (4pt) so that page breaks will not interfere too much. If you want columns, fine, just hook in some code that typesets the final columns as tables. In the end you will have an ugly looking pdf file but by feeding it into `pdftotext` you will get a nicely formatted html file.

For some languages of course encoding issues would show up and there can be all kind of interferences, so eventually the amount of code dealing with it would have accumulated. This is why we don’t follow that route.

An alternative is to use `tex4ht` which does an impressive job for L^AT_EX, and supports ConT_EXt to some extend as well. As far as I know it overloads some code deep down in the kernel which is something ‘not done’ in the ConT_EXt universe if only because we cannot keep control over side effects. It also complicates maintainance of both systems.

In MkIV however, we do have the ability to export the document to a verbose structured so let’s have a look at that.

12.3 Structure

The ability to export to some more verbose format depends on the availability of structural information. As we already tag elements for the sake of tagged

pdf, it was tempting to see how well we could use those tags for exporting to xml. In principle it is possible to use Acrobat Professional to export the content using tags but you can imagine that we get a better quality if we stay within the scope of the producing machinery.

```
\setupbackend[export=yes]
```

This is all you need unless you want to fine tune the resulting xml file. If you are familiar with tagged pdf support in ConTeXt, you will recognize the result. When you process the following file:

```
\setupbackend[export=yes]

\starttext

\startchapter[title=Test]
A paragraph.\par Another paragraph.
\stopchapter

\stoptext
```

You will get a file with the suffix `export` that looks as follows:¹⁷

```
<?xml standalone='yes' encoding='utf-8' ?>

<!-- input filename   : exported-001     -->
<!-- processing date  : 09/08/10 01:00:22 -->
<!-- context version  : 2010.09.05 16:30  -->
<!-- exporter version : 0.10            -->

<document language='en'>
  <section detail='chapter'>
    <sectionnumber>1</sectionnumber>
    <sectiontitle>Test</sectiontitle>
    <sectioncontent>
A paragraph.
      <break/>
Another paragraph.
    </sectioncontent>
  </section>
</document>
```

¹⁷ We will omit the topmost lines in following examples.

It's no big deal to postprocess such a file. In that case one can for instance ignore the chapter number or combine the number and the title. Of course rendering information is lost here. However, sometime it makes sense to export some more details. Take the following table:

```
\starttext

\bTABLE
  \bTR \bTD test 1.1 \eTD \bTD[ny=2] test 1.2 \eTD \eTR
  \bTR \bTD test 2.1 \eTD                                     \eTR
  \bTR \bTD test 3.1 \eTD \bTD test 3.2                    \eTD \eTR
  \bTR \bTD test 4.1 \eTD \bTD                             \eTD \eTR
  \bTR \bTD[nx=2,align=flushright] test 5.1 \eTD \eTR
\eTABLE

\stoptext
```

Here we need to preserve the span related information as well as cell specific alignments as for tables this is an essential part of the structure.

```
<document language='en'>
  <table>
    <tablerow>
      <tablecell align='flushleft'>test 1.1 </tablecell>
      <tablecell align='flushleft' rows='2'>test 1.2 </tablecell>
    </tablerow>
    <tablerow>
      <tablecell align='flushleft'>test 2.1 </tablecell>
    </tablerow>
    <tablerow>
      <tablecell align='flushleft'>test 3.1 </tablecell>
      <tablecell align='flushleft'>test 3.2 </tablecell>
    </tablerow>
    <tablerow>
      <tablecell align='flushleft'>test 4.1 </tablecell>
      <tablecell></tablecell>
    </tablerow>
    <tablerow>
      <tablecell align='flushright' columns='2'>test 5.1 </tablecell>
    </tablerow>
  </table>
</document>
```

The tabulate mechanism is quite handy for regular text especially when the

content of cells has to be split over pages. As each line in paragraph in a tabulate becomes a cell, we need to reconstruct the paragraphs.

```

\starttext

\starttabulate[|l|p|r|]
  \NC zero \NC line one \par line two \par line three \NC 0 \NC \NR
% \NC one \NC \input zapf \par \input zapf \NC 1 \NC \NR
  \NC two \NC before \type {connect} \par after \NC 2 \NC \NR
  \NC three \NC before \type {connect} after \NC 3 \NC \NR
  \NC four \NC before \break inbetween \par after \NC 4 \NC \NR
\stoptabulate

\stoptext

```

This becomes:

```

<document language='en'>
  <tabulate>
    <tabulaterow>
      <tabulatecell align='flushleft'>zero</tabulatecell>
      <tabulatecell>line one
      <break/>
line two
      <break/>
line three</tabulatecell>
      <tabulatecell align='flushright'>0</tabulatecell>
    </tabulaterow>
    <tabulaterow>
      <tabulatecell align='flushleft'>two</tabulatecell>
      <tabulatecell>before <verbatim>connect</verbatim>
      <break/>
after</tabulatecell>
      <tabulatecell align='flushright'>2</tabulatecell>
    </tabulaterow>
    <tabulaterow>
      <tabulatecell align='flushleft'>three</tabulatecell>
      <tabulatecell>before <verbatim>connect</verbatim> after</tabulatecell>
      <tabulatecell align='flushright'>3</tabulatecell>
    </tabulaterow>
    <tabulaterow>
      <tabulatecell align='flushleft'>four</tabulatecell>
      <tabulatecell>before inbetween
      <break/>

```

```

after</tabulatecell>
  <tabulatecell align='flushright'>4</tabulatecell>
</tabulaterow>
</tabulate>
</document>

```

The `<break/>` elements are injected automatically between paragraphs. We could tag each paragraph individually but that does not work that well when we have for instance a quotation that spans multiple paragraphs (and maybe starts in the middle of one). An empty element is not sensitive for this and is still a signal that vertical spacing is supposed to be applied.

12.4 The implementation

We implement tagging using attributes. The advantage of this is that it does not interfere with typesetting, but a disadvantage is that not all parent elements are visible. When we encounter some content, we're in the innermost element so if we want to do something special, we need to deduce the structure from the current child. This is no big deal as we have that information available at each child.

The first implementation just flushed the xml on the fly (i.e. when traversing the node list) but when I figured out that collapsing was needed for special cases like tabulated paragraphs this approach was no longer valid. So, after some experiments I decided to build a complete structure tree in memory¹⁸. This permits us to handle situations like the following:

```

\starttext

\startitemize[n]
  \startitem one \stopitem
  \startitem two \stopitem
\stopitemize

\startitemize[packed,a]
  \startitem \quote{one} \stopitem
  \startitem \quote{two} \stopitem
\stopitemize

\stoptext

Here we get:

```

¹⁸ We will see if this tree will be used for other purposes in the future.

```

<document language='en'>
  <itemgroup detail='itemize' symbol='n'>
    <item>
      <itemtag>1.</itemtag>
      <itemcontent>one</itemcontent>
    </item>
    <item>
      <itemtag>2.</itemtag>
      <itemcontent>two</itemcontent>
    </item>
  </itemgroup>
  <itemgroup detail='itemize' packed='yes' symbol='a'>
    <item>
      <itemtag>a.</itemtag>
      <itemcontent><delimited detail='quote'>'one'</delimited></itemcontent>
    </item>
    <item>
      <itemtag>b.</itemtag>
      <itemcontent><delimited detail='quote'>'two'</delimited></itemcontent>
    </item>
  </itemgroup>
</document>

```

The `symbol` and `packed` attributes are first seen at the `itemcontent` level (the innermost element) so when we flush the `itemgroup` element's attributes we need to look at the child elements (content) that actually carries the attribute.¹⁹

I already mentioned collapsing. As paragraphs in a tabulate get split in cells, we encounter a mixture that cannot be flushed sequentially. However, as each cell is tagged unique we can append the lines within a cell. Also, as each paragraph gets a unique number, we can add breaks before a new paragraph starts. Collapsing and adding breakpoints is done at the end, and not per page, as paragraphs can cross pages. Again, thanks to the fact that we have a tree, we can investigate content and do this kind of manipulations.

Moving data like footnotes are somewhat special. When notes are put on the page (contrary to for instance end notes) the so called 'insert' mechanism is used where their content is kept with the line where it is defined. As a result we see them end up instream which is not that bad a coincidence. However, as in MkIV notes are built on top of (enumerated) descriptions, we need to distinguish them somehow so that we can cross reference them in the export.

¹⁹ Only glyph nodes are investigated for structure.

```
\starttext
```

```
\startchapter[title=Notes]
```

```
test \footnote[a]{note a}
```

```
test \footnote[b]{note b}
```

```
\stopchapter
```

```
\stoptext
```

Currently this will end up as follows:

```
<document language='en'>
  <section detail='chapter'>
    <sectionnumber>1</sectionnumber>
    <sectiontitle>Notes</sectiontitle>
    <sectioncontent>
test<descriptionsymbol detail='footnote' insert='1'>1</descriptionsymbol>
test<descriptionsymbol detail='footnote' insert='2'>2</descriptionsymbol>
      <description detail='footnote'>
        <descriptiontag insert='1'>1 </descriptiontag>
        <descriptioncontent>note a</descriptioncontent>
      </description>
      <description detail='footnote'>
        <descriptiontag insert='2'>2 </descriptiontag>
        <descriptioncontent>note b</descriptioncontent>
      </description>
    </sectioncontent>
  </section>
</document>
```

Graphics are also tagged and the image element reflects the included image.

```
\starttext
```

```
\placefigure
```

```
[here] [fig:cow]
```

```
{It looks like a cow.}
```

```
{\externalfigure[cow.pdf]}
```

```
\stoptext
```

If the image sits on another path then that path shows up in an attribute and when a page other than 1 is taken from the (pdf) image, it gets mentioned as well.

```
<document language='en'>
  <float detail='figure' reference='fig:cow'>
    <floatcontent><image name='cow.pdf'></image></floatcontent>
    <floatcaption>
      <floattag>Figure 1</floattag>
      <floattext>It looks like a cow.</floattext>
    </floatcaption>
  </float>
</document>
```

Cross references are another relevant aspect of an export. In due time we will export them all. It's not so much complicated because all information is there but we need to hook some code into the right spot and making examples for those cases takes a while as well.

```
\setupinteraction[state=start]

\starttext

\startchapter[title=One,reference=alpha]
  In \in{chapter}[beta] ...
\stopchapter

\startchapter[title=Two,reference=beta]
  In \in{chapter}[alpha] ...
\stopchapter

\stoptext
```

We export references in the the ConTeXt specific way, so no interpretation takes place.

```
<document language='en'>
  <section detail='chapter' reference='alpha'>
    <sectionnumber>1</sectionnumber>
    <sectiontitle>One</sectiontitle>
    <sectioncontent>
In <link reference='beta' location='aut:2'>chapter 2</link> ...
    </sectioncontent>
  </section>
```



```

<section detail='chapter' reference='beta'>
  <sectionnumber>2</sectionnumber>
  <sectiontitle>Two</sectiontitle>
  <sectioncontent>
In <link reference='alpha' location='aut:1'>chapter 1</link> ...
  </sectioncontent>
</section>
</document>

```

As Con \TeX t has an integrated referencing system that deals with internal as well as external references, url's, special interactive actions like controlling widgets and navigations, etc. and we export the raw reference specification as well as additional attributes that provide some detail.

```

\setupinteraction[state=start]

\useurl [pragma] [www.pragma-ade.com]

\starttext

\startparagraph
  You can visit \goto{pragma}{url(www.pragma-ade.com)}.
\stopparagraph

\startparagraph
  You can visit \goto{pragma}{url(pragma)}.
\stopparagraph

\stoptext

```

Of course, when postprocessing the exported data, you need to take these variants into account.

```

<document language='en'>
  <paragraph>You can visit <link reference='url(www.pragma-ade.com)' url='www.pra
  <paragraph>You can visit <link reference='url(pragma)' url='www.pragma-ade.com'
</document>

```

12.5 Math

Of course there are limitations. For instance \TeX 'ies doing math might wonder if we can export formulas. To some extent the export works quite well.

```
\starttext
```

Is it $e = mc^2$ maybe:

```
\startformula
```

```
m = \frac{\sqrt{e}}{c}
```

```
\stopformula
```

```
\stoptext
```

This results in the usual rather verbose presentation MathML:

```
<document language='en'>
```

```
Is it
```

```
<math>
```

```
<mrow>
```

```
<mi></mi>
```

```
<mo>=</mo>
```

```
<mi></mi>
```

```
<msup>
```

```
<mi></mi>
```

```
<mn>2</mn>
```

```
</msup>
```

```
</mrow>
```

```
</math>
```

```
maybe:
```

```
<formula>
```

```
<formulacontent>
```

```
<math>
```

```
<mrow>
```

```
<mi> </mi>
```

```
<mo>=</mo>
```

```
<mrow>
```

```
<mfrac>
```

```
<mrow>
```

```
<mrow>
```

```
<mo>√ </mo>
```

```
<mroot>
```

```
<mi></mi>
```

```
</mroot>
```

```
</mrow>
```

```
</mrow>
```

```
</mrow>
```

```

        <mi> </mi>
      </mrow>
    </mfrac>
  </mrow>
</mrow>
</math>
</formulacontent>
</formula>
</document>

```

More complex math (like matrices) will be dealt with in due time as for this and Aditya and I have to take tagging into account when we revision the relevant code as part of the MkIV cleanup and extensions. It's not that complex but it makes no sense to come up with intermediate solutions.

Display verbatim is also supported. In this case we tag individual lines.

```

\starttext

\starttyping
line one
line two
\stoptyping

\stoptext

```

The export is not that spectacular:

```

<document language='en'>
  <verbatimblock detail='typing'>
    <verbatimline>
line one
    </verbatimline>
    <verbatimline>
line two
    </verbatimline>
  </verbatimblock>
</document>

```

A rather special case are marginal notes. We do tag them because they often contain usefull information.

```

\starttext

```

```

\startparagraph
  test \inleft{left 1} test
\stopparagraph

\margintitle{left 2}

\startparagraph
  test test
\stopparagraph

\startparagraph
  \inrightmargin{\slanted{right 1}}test
\stopparagraph

\stoptext

```

The output is currently as follows:

```

<document language='en'>
  <paragraph><margintextblock detail='left'>left 1</margintextblock> test
test</paragraph>
  <paragraph>test test</paragraph>
  <paragraph><margintext detail='inrightmargin'> right 1</margintext>
</paragraph></document>

```

However, this might change in future versions.

12.6 Formatting

The output is somewhat formatted. The extra run time needed for this (actually, quite some of the code is related to this) is compensated by the fact that inspecting the result becomes more convenient. Each environment has one of the properties `inline`, `mixed`, and `display`. A `display` environment gets newlines around it and an `inline` environment none at all. The `mixed` variant does something in between. In the following example we tag some user elements, but you can as well influence the built in ones.

```

\setelementnature[display][display]
\setelementnature[inline] [inline]
\setelementnature[mixed] [mixed]

\starttext

```

```

\startelement[display]
  \startelement[inline]
    test
  \startelement[display]
    test
  \stopelement
    test
\stopelement

```

```
\stoptext
```

This results in:

```

<document language='en'>
  <display>
<inline>test <display> test test</display></inline>
  </display>
</document>

```

Keep in mind that elements have no influence on the typeset result apart from introducing spaces when used used this way (this is not different from other \TeX commands). In due time the formatting might improve a bit but at least we have less change ending up with those megabyte long one–liners that some applications produce.

12.7 A word of advise

In (for instance) html class attributes are used to control rendering driven by stylesheets. In $\text{Con}\TeX$ t you can often define derived environments and their names will show up in the detail attribute. So, if you want control at that level in the export, you'd better use the structure related options built in $\text{Con}\TeX$ t, for instance:

```

\definehead[specialsection][section]

\starttext

\startsection[title=Normal section]
  normal
\stopsection

```

```

\startspecialsection[title=Special section]
  special
\stopspecialsection

\stoptext

```

This gives two different sections:

```

<document language='en'>
  <section detail='section'>
    <sectionnumber>1</sectionnumber>
    <sectiontitle>Normal section</sectiontitle>
    <sectioncontent>
normal
    </sectioncontent>
  </section>
  <section detail='specialsection'>
    <sectionnumber>2</sectionnumber>
    <sectiontitle>Special section</sectiontitle>
    <sectioncontent>
special
    </sectioncontent>
  </section>
</document>

```

12.8 Conclusion

It is an open question if such an export is useful. Personally I never needed a feature like this and there are several reasons for this. First of all, most of my work involves going from (often complex) xml to pdf and if you has xml as input, you can also produce html from it. For documents that relate to ConT_EXt I don't need it either because manuals are somewhat special in the sense that they often depend on showing something that ends up on paper (or its screen counterpart) anyway. Loosing the makeup also renders the content somewhat obsolete. But this feature is still a nice proof of concept anyway.

Marking Proof-sheets in Publishing Practice and Its Implementation in the T_EX System

Označování korekturních obsahů v nakladatelské praxi a jeho implementace v systému T_EX

TOMÁŠ HÁLA

Abstract: This paper deals with ways of marking proof-sheets in publishing practice. Four possible solutions are shown and discussed. Three of them are based on existing macros (page style `\headings`), or packages (`fancyhdr.sty`, `zwpagelayout.sty`), the fourth one is original and specific, and contains new style for L^AT_EX – `thproof.sty`.

Keywords: proof-sheets, proof-reading, publishing practice, L^AT_EX, styles, `thproof.sty`

Abstract: Tento článek se zabývá možnostmi označování korekturních obsahů v nakladatelské praxi a představuje čtyři taková řešení. Tři z nich využívají existující makropříkazy nebo styly (`\headings`, `fancyhdr.sty`, `zwpagelayout.sty`). Čtvrté je původní a specializované řešení pomocí nového stylu pro L^AT_EX – `thproof.sty`.

Klíčová slova: korektury, korekturní obsahy, nakladatelská praxe, L^AT_EX, styly, `thproof.sty`

1. Introduction

Even the most modern technologies cannot prevent various mistakes in text processing. Therefore, proofreading is a very important and often priceless activity.

Basic review of proofreading topic, i.e. itemisation of proofreading according to time and content points of view, organisational and methodical ways of proofreading and a comparison of ‘classical’ and modern procedures has been published earlier (HÁLA, 2002). Of these topics, itemisation proofreading will be mentioned in this paper.

2. Itemisation of proofreading

From the time point of view (POP, FLÉGR A POP, 1989) proofreading consists of four basic parts:

- internal galley
- author's galley
- internal page
- author's page

Internal proofreading covers all editing and typesetting corrections carried out in composing room, while author's proofreading is performed by the author himself/herself, or – in some specific cases – by an editor.

From the content point of view, we can distinguish:

editor's proof – verifies documentary part of text;

language proof – eliminates orthographic mistakes, wrong punctuation, misspellings;

proof of style – is connected with editing activities;

proof of formal matters – checks references, numbers of figures and tables, their numbering vs. references in the text; unification of proper emphasizing style, and of different styles used by various authors;

typographic proof – checks proper typefaces, symbols, indentation, placing of objects, typesetting of tables, etc.;

graphical – looks into colour management, colourfulness, contrast, backgrounds, etc.;

technical proof – deals with pagenumbers, signature imposition, technical quality for printing, etc.

Each company, performing larger number of jobs – whether publishing house or (typo)graphical studio – has to keep out of troubles caused by disordered proof-sheets.

Therefore, proof-sheets are usually marked not only by name of the job, but also by additional pieces of information:

- date and time (of printing, usually)
- number of the job
- type of proofreading, including ordinal number
- typesetter's name and signature
- proofreader's name and signature
- author's/client's signature (only for authors' proofreading)

3. Review of solutions based on existing methods

3.1. Page style headings

The principle of page layout is based on definitions prepared in the format \LaTeX . Two basic page styles `empty` and `plain` are extended by page styles `heading` and `myheading` which are included as a part of pre-defined classes, e.g. `article.cls`, `book.cls`.

Page styles are activated by macro (`\pagestyle{headings}`) in the preamble of a document for the whole document, or by `\thispagestyle{headings}` if the property is applied only to one concrete page.

```
\pagestyleheadings
\begindocument
\markright\footnotesize \TeX --
%Brejlov-- 2nd, author's page proofsheet,
September 16, 2010, --
testing-headings -- p. 1
```

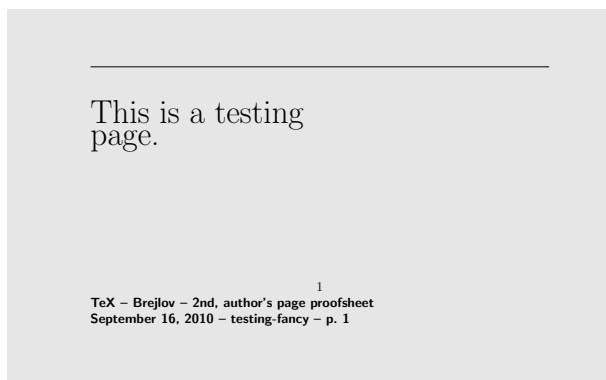
The head line contains only one line, for longer text the user has to use some advanced solution.

With this style no extra additions are needed for changing the layout of a document. Setting a layout in this way does not, however, belong to popular methods. This is proved by the very existence of the package `fancyhdr.sty` which has been created for simplifying L^AT_EX users work.

3.2. Package `fancyhdr.sty`

Package `fancyhdr.sty` (OOSTRUM, 2005) makes the configuration of the page style more simple. The following figure shows the use of commands `fancyfoot` and `fancyhead` parametrised by positions of elements.

```
\fancyfoot[OC,EC]{\thepage}
\fancyfoot[OL,ER]{\TeX\ -- Brejlov -- 2., autorské stránkové
korektury\\today\ -- \jobname\ --
str. \thepage}
```



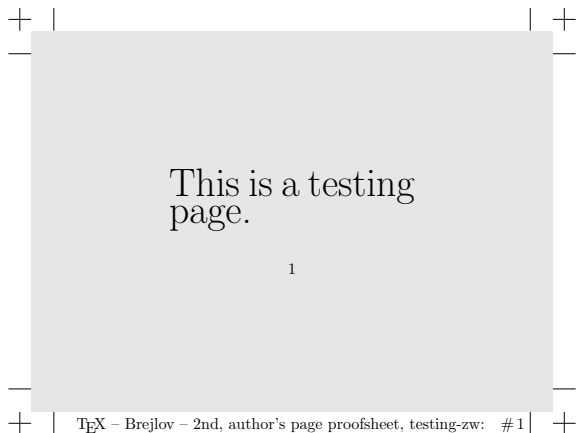
The package `fancyhdr.sty` seems to be very useful for user's styling, however, for joining the design information with proof-sheet marking one has to apply the proof-sheet marking settings to each change of style, i.e. to each use of the corresponding macros which might cause some inconsistencies.

3.3. The package for the page layout `zwpagelayout.sty`

This package has been written by WAGNER (2008) as a very complex and useful solution for all who want to prepare layout definitions comfortably.

The package is useful also for preparation of book covers. It contains a tool for placing the marks showing the position of spine, flaps, frames, etc. It also supports crop titles:

```
\usepackage[a6,Landscape, croptitle=\TeX\ --\ Brejlov\ --\ 2.{,}\
  autorské\ stránkově\ korektury]{zwpagelayout}
```



However very useful is this tool, there are some limitations. First, the command `\usepackage` ignores spaces between words in options, so the user has to write ‘backslashed’ spaces instead of normal ones¹, or to enclose the text with an extra pair of braces.

Moreover, the crop title option is joined with the pair `cropmarks/nocropmarks` option so that displaying the crop title is excluded when the typesetter decides to suppress crop marks.

Limiting is also the length of crop title value: only one line is permitted² so that the complete information cannot always be displayed.

¹Similar approach require commas: they work as delimiters between options, so the proper writing is somewhat tricky, e.g. `{,}`.

²The thing is that the command `\\` is suppressed in options of `\usepackage`.

4. Specialised solution: style `thproof.sty`

This package is the result of a development inspired by author's everyday needs. The first version of this style has been prepared in 2001 for internal use in the publishing house KONVOJ, the latest version (1.06, 2004) has been published at www.konvoj.cz/styles/korek.

The basic requirements were: (a) independence of marking up and layout definitions, (b) portability between L^AT_EX 2.09, L^AT_EX 2_ε, formats cslatex, pdfcslatex, etc.

The work matured in `thproof.sty` (successor of `korek.sty`); the name of this style has been chosen so as not to be in conflict with the style `proof.sty` (TATSUTA, 1990), which improves some mathematical matters.

4.1. Implementation

This solution is based on redefining the low-level T_EX structures. The macro `\@outputpage` encloses among other things the call of macro `\shipout` sending the prepared vertical box with contents of the page to the output. In this vertical box, the extra part has been added (see page ??; for transparent programming, it has been expressed by one macro (`thproof@markspace`) defined in advance.

For preparing the marking structure, the L^AT_EX environment `picture` has been used, as follows:

```
\def\thproof@markspace{\unitlength1cc
\begin{picture}(0,0)
\put(\thproof@x,\thproof@y){%
\parbox{\hsize}{\thproof@font\thproof@text}}
\end{picture}}
```

4.2. User interface

The following commands have been defined for users:

#	name of the macro	and its parameters	note
1 & 2	<code>\thproofPosition</code>	<code>{0}{-38}</code>	a position, where the marking text will be placed; default unit is 1cc
3	<code>\thproofCompany</code>	<code>{\TeXpe... 2010}</code>	name of a (publishing) company
4	<code>\thproofJobNo</code>	<code>{1/2010}</code>	internal job number in a (publishing) company
5	<code>\thproofJob</code>	<code>{Brejlov}</code>	job name or client's name
6	<code>\thproofAuthor</code>	–	author's proofreading
6	<code>\thproofHome</code>	–	internal proofreading
7	<code>\thproofGalley</code>	<code>{1st}</code>	galley proofreading and its number
7	<code>\thproofPage</code>	<code>{2nd}</code>	page proofreading and its number
–	<code>\thproofImprimatur</code>	–	produces a stamp for authorisation
–	<code>\thproofEnd</code>	–	no marking texts will be printed

On the preceeded pages the markuping text has been generated by

```
\thproofPosition{0}{0}
\thproofCompany{\TeXperience 2010}
\thproofAuthor
\thproofJobNo{1/2010}
\thproofJob{Brejlov}
\thproofGalley{1.}
```

and

```
\thproofHome \thproofPage{2.}
```

respectively.

One can also use the shorter version via general setting command:

```
\thproofSettings{0}{-65}{\TeX{p}erience 2010}
{1/2010}{Brejlov}
{H}{G}{1}
```

The order of the first seven parameters corresponds to the table above, the eighth one is the number of a proofsheets.

For the final version of a document the supplier very often requires the client's signature which expresses the client's authorisation of the document before printing.

Macro `\thproofImprimatur`³ generates the stamp containing the space for the date and client's signature.

³Imprimatur is a Latin word with meaning *let it be printed*.

TeXperience 2010 JOB: 1/2010– p. 306	
IMPRIMATUR	
Date 16. 9. 2010	Signature

When marking the text is not necessary, e.g. for the printing version, it can be switched off using macro `\thproofEnd`.

5. Conclusion

The proper marking proof-sheets is necessary because of organisational reasons. In this article, three possible solutions are shown (`\headings`, packages `fancyhdr.sty`, `zwpagelayout.sty`). None of them – even if they are based on existing styles, partly very sophisticated ones – do not live to expectations of the author of this article – some technical restrictions obstruct comfortable use for proof-sheet marking.

Therefore the main point of this work was to present the new way of solving mentioned problems. The L^AT_EX style `thproof.sty` seems, from the user's point of view, to be fairly simple and comfortable solution. Macros covering each part of proof-sheet description have been shown, as well as one general setting command.

This style is maintained by author and tested in his publishing practice.

References

- HÁLA, TOMÁŠ. Korektury a korektoři v 21. století. [Proofreading and proofreaders in the 21st century]. *Zpravodaj Československého sdružení uživatelů T_EXu*, 2002, 3–4, s. 152–159. (ISSN 1211-6661.)
- VAN OOSTRUM, PIET. `fancyhdr.sty`. In FEUERSTACK, THOMAS; BERRY, KARL; KOCH, RICHARD; LOTZ, MANFRED (EDS.) *T_EX Collection, August 2010 [DVD]*. Berlin: Lehmanns Media, c2005. (ISBN 978-3-86541-401-4.)
- POP, PAVEL; FLÉGER, JINDŘICH; POP, VLADIMÍR. *Sazba I Ruční sazba [Typesetting I]*. 2. vyd. Praha: SPN, 1989. 188 s.
- TATSUTA, MAKOTO. `proof.sty`. In FEUERSTACK, THOMAS; BERRY, KARL; KOCH, RICHARD; LOTZ, MANFRED (EDS.) *T_EX Collection, August 2010 [DVD]*. Berlin: Lehmanns Media, c1990–2005. (ISBN 978-3-86541-401-4.)
- WAGNER, ZDENĚK. `zwpagelayout.sty`. In FEUERSTACK, THOMAS; BERRY, KARL; KOCH, RICHARD; LOTZ, MANFRED (EDS.) *T_EX Collection, August 2010 [DVD]*. Berlin: Lehmanns Media, c2008. (ISBN 978-3-86541-401-4.)

T_EXperience 2010 JOB: 1/2010– p. 307	
IMPRIMATUR	
Date 16. 9. 2010	Signature

Integration of thproof@markspace into @outputpage (the source code has been taken from latex.ltx):

```

\def\@outputpage{%
\begin@group          % the \endgroup is put in by \aftergroup
\let \protect \noexpand
\@resetactivechars
\@parboxrestore
\shipout \vbox{%
\set@typeset@protect
\aftergroup \endgroup
\aftergroup \set@typeset@protect
% correct? or just restore by ending
% the group?
}
\if@specialpage
\global\@specialpagefalse\@nameuse{ps@\@specialstyle}%
\fi
\thproof@markspace %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\if@twoside
\ifodd\count@z@ \let\@thehead\@oddshead \let\@thefoot\@oddsfoot
\let\@themargin\@oddsidemargin
\else \let\@thehead\@evenshead
\let\@thefoot\@evenfoot \let\@themargin\@evensidemargin
\fi
\fi
\reset@font
\normalize
\baselineskip\z@skip \lineskip\z@skip \lineskiplimit\z@
\@begin@div
\vskip \topmargin
\moveright\@themargin \vbox {%
\setbox\@tempboxa \vbox to\headheight{%
\vfil
\color@hbox
\normalcolor
\hb@xt@\textwidth {%
\let \label \gobble
\let \index \gobble
\let \glossary \gobble %% 21 Jun 91
\@thehead
}%
\color@endbox
}% %% 22 Feb 87
\dp\@tempboxa \z@
\box\@tempboxa
\vskip \headsep
\box\@outputbox
\baselineskip \footskip
\color@hbox
\normalcolor
\hb@xt@\textwidth{%
\let \label \gobble
\let \index \gobble %% 22 Feb 87
\let \glossary \gobble %% 21 Jun 91
\@thefoot
}%
\color@endbox
}%
}
\global \colht \textheight
\stepcounter{page}%
\let\firstmark\botmark
}

```

Tomáš Hála

1. Department of Informatics, Faculty of Economy, Mendel University, Brno,
Zemědělská 1, 613 00 Brno, Czech Republic
2. Publishing house KONVOJ, spol. s r. o. (Ltd.), Brno, Czech Republic
email: thala@mendelu.cz, konvoj@konvoj.cz

T_EXperience 2010	
JOB: 1/2010– p. 308	
IMPRIMATUR	
Date 16. 9. 2010	Signature

Fonts with Complex OpenType Tables

Fonty se složitými tabulkami ve formátu OpenType

KAREL PÍŠKA

Abstract: The paper presents development of complex OpenType fonts. The sample fonts cover Czech and Georgian handwriting with numerous letter connections.

At the beginning, general principles of “advanced typography” are shown – complex metric data represented by OpenType tables (GSUB and GPOS) – and compared them with the ligature and kerning tables in METAFONT.

Then the history of the OpenType font production is described – approaches, tools and techniques. Crucial problems, critical barriers, attempts and ways how to reach successful solutions, are discussed and several tools for font creating, testing, debugging and conversions between various text and binary formats are demonstrated. Among these tools are, for example, AFDKO, VOLT, FontForge, TTX, Font-TTF. Their features, advantages, disadvantages, and also cases of possible incompatibilities (or maybe errors) are illustrated.

Finally, using the OpenType fonts in the \TeX world applications are presented: $X_{\text{f}}\TeX$ and $\text{Lua}\TeX$ (CON $\text{T}_{\text{E}}\text{X}$ MkIV), the programs allowing to read and process OpenType fonts directly.

Key words: font, font production, Unicode, OpenType, GSUB, GPOS; AFDKO, VOLT, FontForge, TTX, Font-TTF; \TeX , METAFONT, TFM, $X_{\text{f}}\TeX$, CON $\text{T}_{\text{E}}\text{X}$, $\text{Lua}\TeX$.

Abstrakt: Článek popisuje vývoj složitých fontů ve formátu OpenType v letech 2009–2010. Ukázky zahrnují český a gruzínský rukopisný font s mnohočetnými spojeními mezi sousedními písmeny.

Na začátku ukážeme obecné principy „pokročilé typografie“: složitá metrická data reprezentovaná tabulkami GSUB a GPOS v OpenType, které porovnáme s tabulkami ligatur a kerningů v METAFONTU.

Potom popíšeme historii tvorby OpenTypového fontu: postupy, nástroje a techniky. Probereme klíčové problémy, závažné překážky, pokusy a způsoby řešení k dosažení úspěšného výsledku. Předvedeme několik nástrojů pro tvorbu, testování a ladění fontů a konverze mezi různými textovými a binárními formáty jejich reprezentace. Jsou to např. AFDKO, VOLT, FontForge, TTX, Font-TTF. Budeme ilustrovat jejich vlastnosti, výhody, nevýhody, také i případy možných nekompatibilit (anebo možných chyb).

Nakonec předvedeme použití OpenTypových fontů v rámci \TeX u: $X_{\text{f}}\TeX$ a $\text{Lua}\TeX$ (CON $\text{T}_{\text{E}}\text{X}$ MkIV) jsou programy dovolující číst a zpracovávat fonty OpenType přímo, tj. bez tradičních metrik TFM.

Klíčová slova: font, tvorba fontů, Unicode, OpenType, GSUB, GPOS; AFDKO, VOLT, FontForge, TTX, Font-TTF; T_EX, METAFONT, TFM, X_ƎT_EX, CON_TE_XT, Lua_TE_X.

1. Introduction

The presented fonts are successors of the METAFONT fonts designed in 1997–98 by Olšák [2] and Píška [3]. Last year (2009) it was not possible for me to create a complete font with OpenType tables that would work properly. This year, finally positive results have been reached. The current article can be considered as a report summarizing my recent studies, experiments and experiences for dialogs and future collaboration with involved people. My main direction prefers the usage of fonts within T_EX based software providing Unicode and OpenType support – X_ƎT_EX [9] and CON_TE_XT/Lua_TE_X [11]. For OpenType the abbreviation “OT” will also be used in the article.

2. Advanced typography

Under “advanced typography” not only so called OpenType font technologies but also our good “old” T_EX&METAFONT capability providing sophisticated word-processing can be assumed.

2.1. T_EX & METAFONT – clear and clean

In fact, advanced typography with METAFONT and T_EX has been available for T_EX users for many years. METAFONT contains powerful tools like generalized ligatures together with boundary characters [1]:

```
.mf: ligtable % produces % .tfm/.pl
a : b |=:| c ; % acb /LIG/ 1
a : b |=:|> c ; % acb /LIG/> 2
a : b |=:|>> c ; % acb /LIG/>> 3
a : b =:| c ; % cb LIG/ 4
a : b =:|> c ; % cb LIG/> 5
a : b |=: c ; % ac /LIG 6
a : b |=:> c ; % ac /LIG> 7
a : b =: c ; % c LIG 8
```

where

1. retains both *a* and *b*, inserts *c* between: *acb*

2. retains both *a* and *b*, inserts *c* between;
the processing continues after *a*: *acb*
3. retains both *a* and *b*, inserts *c* between;
the processing continues after *c*: *acb*
4. retains *b*, inserts *c* before *b*: *cb*
5. retains *b*, inserts *c* before *b*;
the processing continues after *c*: *cb*
6. retains *a*, inserts *c* after *a*: *ac*
7. retains *a*, inserts *c* after *a*;
the processing continues after *a*: *ac*
8. substitutes both *a* and *b* by *c*.

Boundary characters. The METAFONT and T_EX concept of the “word boundary” (the left and right boundary characters) allows “implicit” processing of the beginning and the end of the word, i.e., a substitution or adjustment of the letters in the “initial” and the “final” position of the word. In METAFONT sources the left boundary characters is denoted by "||:", the right boundary character must be introduced as the “real” character using the "boundarychar *code*";" assignment.

These facilities allow to apply substitution and positioning rules with some restrictions: only the pair of two adjacent characters can be processed, it is impossible to look ahead for longer sequence in a simple way; the maximum of glyphs in one font is 256. However, definitions of ligatures and kernings in METAFONT and then in TFM, and also the processing algorithm in T_EX are clear and clean. The actual position in the input stream and how to find the next rule from T_EX metrics tables that have to be applied are always known.

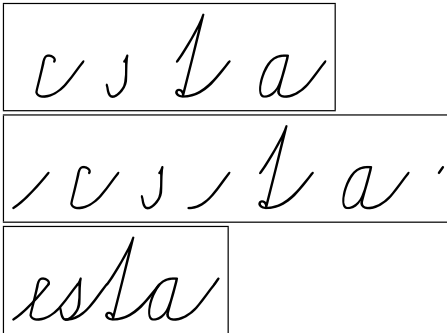
Abilities of METAFONT and T_EX will be demonstrated by two short samples. Primarily, by default, the Latin (Czech) letters are in the “medial” form, without connecting strokes. Then the T_EX&MF “machinery” joins the adjacent letters in words and adjusts the letters in the initial and final positions. The letter ‘e’ is preceded by one of the front-end strokes, ‘s’ and ‘t’ are joined by the corresponding inter-letter connecting stroke, and, finally, the last letter in the word is closed by the ending stroke. METAFONT defines several initial, medial and final strokes (depending on concerned letters), for example:

```
% left deflected end of character
beginchar(3, 6u#, 7u#, 0);
  draw (0,0){(3,2)}..{sklon2}(6,6);
endchar;
% shorter convex stroke for the pairs st,...
beginchar(6, 3u#, 7u#, 0);
  draw (-4,0){right}..{sklon2}(3,6);
endchar;
% right end of character
beginchar(1, .7u#, 7u#, 0);
  draw (0,6)..(.7,7);
endchar;
```

and the `ligtable` instructions

```
ligtable ||: "e" |=:|> 3; % ...
ligtable "s": "t" |=:| 6; % ...
boundarychar:=1;
ligtable "a": rightboundaries;
def rightboundaries =
    1 |=:> 1,
%     .....
enddef;
```

invoke inserting the requested strokes in the left boundary point (3), between ‘s’ and ‘t’ (6), and in the right boundary position (1).



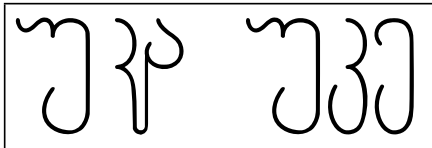
METAFONT cannot process in a single and natural way any character sequences consisting of three or more characters, e.g. triplets like "UN-KAN-AN" and "UN-KAN-EN" in Georgian handwriting.

Depending on the following character ("AN", "EN" or another) the original ("isolated" by default) glyph "KAN" is, or is not, replaced by its modified form:

```
ligtable GR_KAN: GR_AN  =:| GR_kan_;
```

and then it may be joined to the next character by the connecting stroke:

```
ligtable GR_kan_: GR_AN |=:| gr_en_an;
```



But no substitution and no kerning is defined for the pairs "UN-KAN" and "KAN-EN". After processing of the pair consisting of the second and third character and substituting of the second glyph there is no chance to return before the first character it is not possible to adjust the kerning between the first and the second, modified, glyph (left) – while "UN-KAN-EN" (right) needs no substitution

or positioning changes. Two triplets above should be processed differently. It may probably be possible but the solution with METAFONT would not be trivial.

2.2. Advanced typography with OpenType

“Old TrueType fonts” can be “enriched” by adding “Advanced OpenType Typographic Tables” to produce fonts in OpenType format. Since the additional OT tables are common, two different format versions: “new” TTF and OTF will not be discussed.

Each *feature* is defined as a system of subsystems called *lookups*. Any *lookup* is described as a subsystem consisted of substitution and positioning rules. Depending on *script* and *language*, a *feature* may be enabled or disabled. If the *feature* is enabled and some *lookup*, contained in this *feature*, fulfill the given conditions, then the execution of the corresponding operations should be invoked. It is a signal and the real application must be executed by an application program or operating system, e.g., by means of a special library. OpenType introduces substitution (GSUB), positioning (GPOS), and several other tables.

These tables define the set of rules of several types specifying (from OpenType specification [4, 5]):

Glyph substitution (GSUB) rules – Single, Multiple, Alternate, Ligature, Contextual, Chaining contextual, Extension, and Reverse Chaining Single Substitution;

Glyph positioning (GPOS) rules – Single adjustment, Pair adjustment, Cursive attachment, Mark-to-Base attachment, Mark-to-Ligature attachment, Mark-to-Mark attachment, Contextual, Chaining contextual, and Extension positioning.

>From METAFONT entire letters with accents are inherited. Therefore, there is no need to use marks and anchors and operations with them to assemble the complete letters from components (accents, signs, marks, ...) and the Mark positioning rules are not used. On the other hand, the fonts contain hundreds contextual substitution and positioning rules.

First, an OpenType font has to be created properly, using some suitable tools. Secondly, the font must be in agreement with the corresponding software to execute adequate operations according to the rules (instructions) defined in the font.

3. Tools to produce OpenType fonts

3.1. Creating OpenType fonts

Let us assume that Unicode encoded outline fonts have already been encoded, though without OpenType features. These have been generated earlier with Font-

Forge. The aim and task is to produce OpenType, i.e., to enrich the fonts with OT tables.

The fonts used/presented in this paper cover Czech, Georgian and also Armenian handwriting letter repertoire (taught in primary/elementary schools). Opposite to Czech and Georgian writing, the Armenian letters are designed and written in a simple way without no special joiners and there is no necessity to build any OpenType support/facility for Armenian.

Sample of Armenian: *ս ւ զ ր ը սյսյր ու ոսյսսսյր*

There is, however, another problem – to distinguish the adjacent letters.

In the following paragraphs the construction of OT fonts using various tools, namely VOLT, **FontForge** and AFDKO, are shown.

The specification of (binary) OT tables, data formats of the VOLT project files, variants of feature files accepted by AFDKO, FontLab, FontForge may all be different.

3.2. Managing OpenType with VOLT

VOLT (Visual OpenType Layout Tool) [6], free product developed by Microsoft and running only under MS Windows, offers an interactive approach to fill input areas with appropriate parameter values manually in the VOLT project window. Another possibility is writing and modifying source textual files in the VOLT project language. In the VOLT input area one can enter, or in a text editor we have to define glyphs (their names, types and code numbers), glyph groups (glyph sets or glyph lists), context conditions, substitution and positioning rules, and finally, to complete the hierarchy of scripts, languages, features, and lookups; those data can be saved and (re)read. Such method is reasonable and purposeful/meaningful for fonts with several hundreds contextual substitution and positional rules (our font contains about 350 glyphs, more than 600 substitutions and about 50 positionings). Of course, interactive design and especially proofing tools for testing tasks, have been used but the files defining OpenType data have been completed in the VOLT project (VTP) source/exchange format from some tables by scripting and editing texts. VOLT allows to read the font only in TrueType format, imports the VOLT project file, compiles OT data and then generates the font with binary OT tables. That is, VOLT adds OT tables and proofs the features and lookups; it accepts only the fonts with OT tables produced by VOLT, and deletes other OT tables. Moreover, (re)compilation must always be run before testing in the proofing window, even for fonts generated by program VOLT. These fonts embed additionally special tables ‘TSID’, ‘TSIP’, ‘TSIS’, and ‘TSIV’ for proofing.

A general structure of a lookup with substitutions in the VOLT project language (in my symbolic notation) is:

```

DEF_LOOKUP lookup_name lookup_parameters
[ IN_CONTEXT | EXCEPT_CONTEXT
  [ [ LEFT | RIGHT ] glyph_list ]
  ...
  END_CONTEXT
]
AS_SUBSTITUTION
  SUB glyph_list WITH glyph_list END_SUB
[ SUB glyph_list WITH glyph_list END_SUB ]
...
END_SUBSTITUTION

```

It is a sequence of one or more substitution rules and has the common contextual condition. The context may be defined as a compound logical expression. During the evaluation process the glyphs from the given glyph lists before (**LEFT**) or after (**RIGHT**) are compared relative to the current glyph according to their presence (**IN_CONTEXT**) or absence (**EXCEPT_CONTEXT**). Sequences of more **LEFT** and/or **RIGHT** subconditions can constitute left and right chains, their lengths depend of the numbers of the left and right conditions. In nested subexpressions, **IN/EXCEPT_CONTEXT** might be repeated more times, all of them are subsequently evaluated as a logical union.

The *lookup_parameters* contain the instructions for processing like **PROCESS_BASE**, **PROCESS_MARKS**, **ALL**, **DIRECTION LTR** or **DIRECTION RTL**, etc. In our case – **DIRECTION LTR** (“left to right”) – ‘left’ always means ‘before’; similarly ‘right’ and ‘after’ have the same meaning.

Representation of the VOLT Project data allows insertions, and also different rule types may be in one lookup because the VOLT compiler accepts such rules and can compile them when converting the source data into binary OpenType tables. A final binary font then includes more than 100 internal features, numbered **zz01, . . . , zz99, . . .** It means, the higher level of the VOLT project language turns into greater complexity of the compiled product.

The following text demonstrates several complete examples describing the lookups in the VOLT project textual representation as they are present in my source files of the VOLT based fonts.

3.2.1. *Glyphs, scripts, languages and features*

But at first other elements of the VTP will be mentioned. All glyphs must be listed in the glyph definition section, each glyph command must have in the **DEF_GLYPH** its unique name, its ordinal number (index) in the font ID, its **TYPE** (**BASE**, **MARK**, **COMPONENT**, **LIGATURE**), the **UNICODE** number must be present for the Unicode coded glyphs and are missing for the glyphs from Private Use Area (PUA).

Glyphs can be grouped/collected in named groups to address the groups (glyph lists) in rules simply and shortly. DEF_GROUP commands may consist of glyph sequences GLYPH *glyph_name*, glyph ranges, and also other glyph groups, defined elsewhere but without ambiguity.

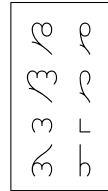
```
DEF_GROUP "czever"
  ENUM RANGE "a" TO "z" GROUP "accver"
  END_ENUM
END_GROUP
```

The names of glyphs and groups must be quoted, e.g., GLYPH "hyphen" or GROUP "czever".

3.2.2. *Single substitution*

Switching the corresponding feature we can invoke the substitution of the letters by their short variants.

```
DEF_LOOKUP "GeorAlt" PROCESS_BASE ALL DIRECTION LTR
AS_SUBSTITUTION
  SUB GLYPH "uni10D3" WITH GLYPH "GR_varD" END_SUB
  SUB GLYPH "uni10DA" WITH GLYPH "GR_varL" END_SUB
  SUB GLYPH "uni10DD" WITH GLYPH "GR_varO" END_SUB
  SUB GLYPH "uni10E0" WITH GLYPH "GR_varR" END_SUB
END_SUBSTITUTION
```



3.2.3. *Ligature substitution*

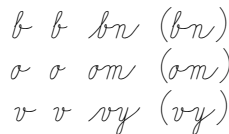
Typical ligatures can be defined by unconditional substitutions.

```
DEF_LOOKUP "liga" PROCESS_BASE ALL DIRECTION LTR
AS_SUBSTITUTION
  SUB GLYPH "comma" GLYPH "comma" WITH GLYPH "quotedblbase" END_SUB
  SUB GLYPH "quoteleft" GLYPH "quoteleft" WITH GLYPH "quotedblleft" END_SUB
  SUB GLYPH "hyphen" GLYPH "hyphen" GLYPH "hyphen" WITH GLYPH "dash" END_SUB
  SUB GLYPH "hyphen" GLYPH "hyphen" WITH GLYPH "minus" END_SUB
END_SUBSTITUTION
```

3.2.4. *Contextual substitution*

Before the letters defined by the right context the letters listed later will be changed to their narrower variants (the unadjusted versions in parenthesis).

```
DEF_LOOKUP "CZEbmnvwy" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
  RIGHT ENUM GLYPH "m" GLYPH "n" GLYPH "ncaron" GLYPH "v" GLYPH "w"
  GLYPH "y" GLYPH "yacute" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
  SUB GLYPH "b" WITH GLYPH "bncolon" END_SUB
  SUB GLYPH "o" WITH GLYPH "onarrow" END_SUB
  SUB GLYPH "oacute" WITH GLYPH "oacutenarrow" END_SUB
  SUB GLYPH "v" WITH GLYPH "vnarrow" END_SUB
  SUB GLYPH "w" WITH GLYPH "wnarrow" END_SUB
END_SUBSTITUTION
```



Following some letters (the left context) the letters “s” and “š” are substituted by their ‘depth’ forms.

```
DEF_LOOKUP "CZEgjqy" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
  LEFT ENUM GLYPH "g" GLYPH "G" GLYPH "j" GLYPH "J" GLYPH "q" GLYPH "Q"
  GLYPH "y" GLYPH "yacute" GLYPH "Y" GLYPH "Yacute" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
  SUB GLYPH "s" WITH GLYPH "sdepth" END_SUB
  SUB GLYPH "scaron" WITH GLYPH "scarondepth" END_SUB
END_SUBSTITUTION
```

3.2.5. Contextual insertion

Between the selected glyphs and their right successors the joining stroke will be inserted.

```
DEF_LOOKUP "CZEjoins_s" PROCESS_BASE ALL DIRECTION LTR
IN_CONTEXT
  RIGHT ENUM GLYPH "m" GLYPH "n" GLYPH "ncaron" GLYPH "t" GLYPH "tcaron"
  GLYPH "y" GLYPH "w" GLYPH "y" GLYPH "yacute" GLYPH "z" GLYPH "zcaron" END_ENUM
END_CONTEXT
AS_SUBSTITUTION
  SUB GLYPH "s" WITH GLYPH "s" GLYPH "joins" END_SUB
  SUB GLYPH "scaron" WITH GLYPH "scaron" GLYPH "joins" END_SUB
  SUB GLYPH "sleft" WITH GLYPH "sleft" GLYPH "joins" END_SUB
  SUB GLYPH "scaronleft" WITH GLYPH "scaronleft" GLYPH "joins" END_SUB
  SUB GLYPH "sdepth" WITH GLYPH "sdepth" GLYPH "joins" END_SUB
  SUB GLYPH "scarondepth" WITH GLYPH "scarondepth" GLYPH "joins" END_SUB
END_SUBSTITUTION
```

3.2.6. Kerning positioning

This lookup operating on glyph pairs defines the kern advances between several FIRST and one SECOND glyph (uni10D6). There are no explicit shifts for the first glyphs, otherwise the second glyph is moved by two values, DX and ADV, for left and ride sides, respectively. In other words, the value DX defines position of the second glyph to the first glyph, whereas the value ADV adjusts the position of the third glyph which follows the pair.

```
DEF_LOOKUP "GEOzen" PROCESS_BASE ALL DIRECTION LTR
AS_POSITION
ADJUST_PAIR
  FIRST ENUM GLYPH "uni10D3" GLYPH "GR_varD" GLYPH "GR_don" GLYPH "uni10D4"
  GLYPH "uni10D5" GLYPH "uni10D6" GLYPH "uni10D7" GLYPH "uni10D8"
  GLYPH "uni10D9" GLYPH "uni10DA" GLYPH "GR_varL" GLYPH "GR__las"
  GLYPH "uni10DD" GLYPH "GR_var0" GLYPH "uni10DF" GLYPH "GR__jan"
  GLYPH "uni10E1" GLYPH "GR__san" GLYPH "uni10E7" GLYPH "uni10E2"
  GLYPH "uni10E3" GLYPH "uni10E4" GLYPH "uni10E6"
  GLYPH "GR__ghan" GLYPH "uni10EA" GLYPH "uni10EF" END_ENUM
```

```

FIRST ENUM GLYPH "uni10D0" GLYPH "uni10D1" GLYPH "GR_ban"
  GLYPH "uni10ED" GLYPH "uni10EE" END_ENUM
FIRST ENUM GLYPH "uni10DC" GLYPH "uni10DE" END_ENUM
FIRST ENUM GLYPH "uni10E8" GLYPH "uni10E9" GLYPH "GR_chin" END_ENUM
FIRST ENUM GLYPH "uni10DB" GLYPH "uni10E5" GLYPH "uni10EB" END_ENUM
FIRST ENUM GLYPH "GR_varR" GLYPH "GR_rae" END_ENUM
SECOND GLYPH "uni10D6"
1 1 BY POS END_POS POS ADV -170 DX -170 END_POS
2 1 BY POS END_POS POS ADV -120 DX -120 END_POS
3 1 BY POS END_POS POS ADV -70 DX -70 END_POS
4 1 BY POS END_POS POS ADV -50 DX -50 END_POS
5 1 BY POS END_POS POS ADV -40 DX -40 END_POS
6 1 BY POS END_POS POS ADV -80 DX -80 END_POS
END_ADJUST
END_POSITION

```

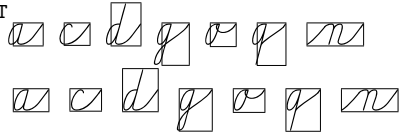
3.2.7. *Left boundary positioning*

If the selected letters are preceded by some non-letter character (its absence in the set "czebeg" marks the word beginning) they will be adjusted

```

DEF_LOOKUP "CZEBegpos" PROCESS_BASE ALL DIRECTION LTR
EXCEPT_CONTEXT LEFT GROUP "czebeg" END_CONTEXT
AS_POSITION
ADJUST_SINGLE
GLYPH "a" BY POS ADV 80 DX 80 END_POS
GLYPH "aacute" BY POS ADV 80 DX 80 END_POS
GLYPH "c" BY POS ADV 80 DX 80 END_POS
GLYPH "ccaron" BY POS ADV 80 DX 80 END_POS
GLYPH "d" BY POS ADV 80 DX 80 END_POS
GLYPH "dcaron" BY POS ADV 80 DX 80 END_POS
GLYPH "g" BY POS ADV 80 DX 80 END_POS
GLYPH "o" BY POS ADV 80 DX 80 END_POS
GLYPH "oacute" BY POS ADV 80 DX 80 END_POS
GLYPH "q" BY POS ADV 80 DX 80 END_POS
END_ADJUST
END_POSITION

```



The letters [a, á, c, č, d, đ, g, o, ó, q] in the medial positions follow the foregoing letters immediately. However, they have some left ‘overshots’ and have to be adjusted if they are in the initial position – when no letter is before them. The condition "EXCEPT_CONTEXT LEFT" is just fulfilled for non-letter glyphs.

3.3. Creating OpenType with FontForge

This subsection illustrates producing OpenType fonts using the files in “OpenType feature files” (FEA) [7], defined by Adobe, and generating the fonts by FontForge [8]. Because all attempts to convert metric data from METAFONT/TFM or VOLT project data failed, the procedure had to be started from scratch again.

Lookups with substitution and positioning rules into textual “feature language” file were rewritten, again “manually” which that the procedure started

with some simple tables with the glyph names and an information about their transformations derived from METAFONT sources; then the files were produced by scripting and modified them with text editors to obtain the required format.

The syntax and lookup structure between VTP and feature file specification differs:

```
lookup lookup_name {  
  [ sub glyph by glyph; ]           #1  
  ...  
  [ sub glyph_list by glyph; ]       #2  
  ...  
  [ sub glyph by glyph_list; ]       #3  
  ...  
  [ sub glyph' glyph_list by glyph; ] #4  
  ...  
  [ sub glyph_list glyph' by glyph; ] #5  
  ...  
  [ sub glyph_list by glyph_list; ]   #6  
  .....  
} lookup_name;
```

In FEA the context is connected with each single rule. In the “feature language” the insertion is not supported, i.e., the character cannot appear at the same time on the left and right side of a substitution rule (opposite to the VOLT project). The rules like

```
sub glypha' glyph_list by glypha glyph;  
sub glypha' glyph_list by glyph glypha;
```

are invalid and unsupported. Another restriction is that one low level lookup must contain a sequence of only one type of substitution rules (only one from type #1 or #2 ... #6, but those rules may be repeated many times). Violating the constrains results in compilation fatal errors. Generally, the sequence between "sub" and "by" may consist of more glyphs to substitute (with apostrophes) and more context elements (without apostrophes) constituting a longer chain and forming a compound conditional expression.

Because of different syntax and structure the commands had to be rebuilt completely. Therefore taking the rule set from the VOLT project each insertion rule had to be divided into two rules and also split some lookups into separate parts. It is necessary to append (in contradiction with the VOLT project) to the font explicitly many new additional intermediate glyphs, the number of glyphs increases from about 350 to 670, and the number of substitution rules increases from about 600 to 850.

VOLT accepts

```
SUB GLYPH "B"  
WITH GLYPH "B" GLYPH "joinc" END_SUB
```

But in FEA

```
sub B' @CZEjoinc by B joinc;
```

is the fatal error.

Extra glyphs `glyph.ini` (absent in METAFONT font and VOLT) for all letters [`A.ini` - `Z.ini`, `a.ini` - `z.ini`] and also for all accented letters had to be added to the font. And every insertion rule must be divided in two rules using `glyph.ini` as the intermediate characters.

```
sub B' @CZEjoin by B.ini;  
sub B.ini by B joinc;
```

Moreover, the two rules above cannot be grouped in one lookup because of their different types. Similarly, we had to add the glyphs `glyph.fin` to solve both left and right boundary processing tasks. The initial and final glyph variants will be located in PUA.

3.3.1. *Single substitution*

```
lookup GeorAlt {  
  sub uni10D3 by GR_varD;  
  sub uni10DA by GR_varL;  
  sub uni10DD by GR_varO;  
  sub uni10E0 by GR_varR;  
} GeorAlt;
```

3.3.2. *Ligature substitution*

```
lookup CZEliga {  
  sub comma comma by quotedblbase;  
  sub quoteleft quoteleft by quotedblleft;  
  sub hyphen hyphen hyphen by dash;  
  sub hyphen hyphen by minus;  
} CZEliga;
```

3.3.3. *Contextual substitution*

```
@CZEbmnvwy = [ m n ncaron v w y yacute ];
```

```
lookup CZEbmnvwy {  
  sub b' @CZEbmnvwy by bnnarrow;  
  sub o' @CZEbmnvwy by onarrow;  
  sub oacute' @CZEbmnvwy by oacutenarrow;  
  sub v' @CZEbmnvwy by vnarrow;  
  sub w' @CZEbmnvwy by wnarrow;  
} CZEbmnvwy;
```

The glyphs to substitute are marked by apostrophes, other glyphs between "sub" and "by" denote the (right) context and the current glyphs will be substituted by the glyphs after "by".

The next example shows the left context.

```
@CZEgjqy = [ g G j J q Y yacute Yacute ];
lookup CZEgjqy {
  sub @CZEgjqy s' by sdepth;
  sub @CZEgjqy scaron' by scarondepth;
} CZEgjqy;
```

3.3.4. Contextual insertion

In FEA it must be redefined as two separate substitutions using intermediate glyphs, their names are ended by ".s".

```
lookup CZEjoins_ss {
  sub s' @CZEjoins by s.s;
  sub scaron' @CZEjoins by scaron.s;
  sub sleft' @CZEjoins by sleft.s;
  sub scaronleft' @CZEjoins by scaronleft.s;
  sub sdepth' @CZEjoins by sdepth.s;
  sub scarondepth' @CZEjoins by scarondepth.s;
} CZEjoins_ss;
lookup CZEjoins_s {
  sub s.s by s joins;
  sub scaron.s by scaron joins;
  sub sleft.s by sleft joins;
  sub scaronleft.s by scaronleft joins;
  sub sdepth.s by sdepth joins;
  sub scarondepth.s by scarondepth joins;
} CZEjoins_s;
```

3.3.5. Kerning positioning

These rules define the adjustments for the glyph pairs. In a more general case the glyphs can be changed by glyph groups.

```
lookup PositGeor {
@GEOzen = [ uni10D6 ];
@GEOzen1 = [ uni10D3 GR_varD GR__don uni10D4
uni10D5 uni10D6 uni10D7 uni10D8
uni10D9 uni10DA GR_varL GR__las
uni10DD GR_var0 uni10DF GR__jan
uni10E1 GR__san uni10E7 uni10E2
uni10E3 uni10E4 uni10E6 GR__ghan
uni10EA uni10EF ];
@GEOzen2 = [ uni10D0 uni10D1 GR__ban uni10ED uni10EE ];
@GEOzen3 = [ uni10DC uni10DE ];
@GEOzen4 = [ uni10E8 uni10E9 GR__chin ];
@GEOzen5 = [ uni10DB uni10E5 uni10EB ];
@GEOzen6 = [ GR_varR GR__rae ];
pos @GEOzen1 @GEOzen -170;
```

```

pos @GEOzen2 @GEOzen -120;
pos @GEOzen3 @GEOzen -70;
pos @GEOzen4 @GEOzen -50;
pos @GEOzen5 @GEOzen -40;
pos @GEOzen6 @GEOzen -80;
} PositGeor;

```

3.3.6. *Left boundary positioning*

The glyphs listed in @CZEleftboundary and @GEOleftboundary will be adjusted by 80 (70) units when the foregoing glyphs are **not** the letter that could be the first one in the word – "ignore" reverses the condition.

```

lookup CZEbegpos {
  ignore pos @czebeg @CZEleftboundary';
  pos @CZEleftboundary' < 80 0 80 0> ;
} CZEbegpos;

```

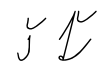



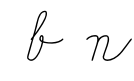















```

lookup LeftPositGeor {
  @GEOleftboundary = [ uni10D1 ];
  @geolet = [ uni10D0 - uni10F0
  GR_varD GR_varL GR_varO GR_varR ];
  ignore pos @geolet @GEOleftboundary';
  pos @GEOleftboundary' < 70 0 70 0>;
} LeftPositGeor;

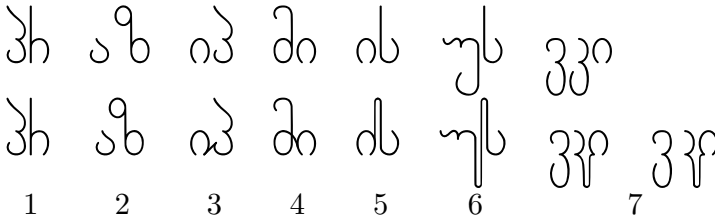
```

3.3.7. *Several comparative examples*

The following examples illustrate several selected cases of rules in their METAFONT, VOLT and FEA implementations. After splitting a FEA rule in two steps we must usually put them in separate lookups depending of the type of the rules.

	št	Pá	Bo	js	bn
p					
c					
s					
					
	1	2	3	4	5

- 1.–3. Various letter pairs are joined by different connecting strokes.
4. The letters “s”/“š” have the modified forms after some letters (g, j, q, y).
5. The letters b, o, v, w are written narrower before m, n, v, y.



1. No connection between letters – no rules defined and applied.
2. Both letters without changes with adjusted kerning.

```

[MF] ligtable GR_AN: GR_ZEN kern kzz#+.2u#;
[VTP] FIRST ...
      FIRST ENUM GLYPH "uni10D0" ... END_ENUM
      ...
      SECOND GLYPH "uni10D6"
      ...
      2 1 BY POS END_POS POS ADV -120 DX -120 END_POS
[FEA] @GEOzen = [ uni10D6 ];
      @GEOzen2 = [ uni10D0 ... ];
      ...
      pos @GEOzen2 @GEOzen -120;

```

3. Letters connected but not modified.

```

[MF] ligtable GR_IN: GR_PAR |=:| gr_in__an;
The connecting stroke inserted after "uni10D8" before "uni10DE".
[VTP] IN_CONTEXT RIGHT GLYPH "uni10DE" END_CONTEXT
      SUB GLYPH "uni10D8" WITH GLYPH "uni10D8" GLYPH "gr_in__an" END_SUB
The new glyph "uni10D8_in__an" must be added and execution in two
steps split in two different lookups (the rule types are not the same).
[FEA] sub uni10D8' uni10DE by uni10D8_in__an;
      sub uni10D8_in__an by uni10D8 gr_in__an;

```

4. Only the first letter changed and connected.

```

[MF] ligtable GR_MAN: GR_IN =:| GR_man_;
[VTP] IN_CONTEXT RIGHT GLYPH "uni10D8" END_CONTEXT
      SUB GLYPH "uni10DB" WITH GLYPH "GR_man_" GLYPH "gr_man__in" END_SUB
In FEA, changing the first glyph and inserting a junction after must be
divided in two steps.
[FEA] sub uni10DB' uni10D8 by GR_man_man__in;
      sub GR_man_man__in by GR_man_ gr_man__in;

```

5. Only the second letter changed and connected.

```

[MF] ligtable GR_IN: GR_SAN |=:| gr_in__san;
[VTP] IN_CONTEXT RIGHT ENUM GLYPH "uni10E1" GLYPH "GR_san_" END_ENUM END_CONTEXT
      SUB GLYPH "uni10D8" WITH GLYPH "uni10D8" GLYPH "gr_in__san" END_SUB
      .....

```

```
IN_CONTEXT LEFT GLYPH "gr_in_san" END_CONTEXT
SUB GLYPH "uni10E1" WITH GLYPH "GR_san" END_SUB
```

```
[FEA] sub uni10D8' uni10E1 by uni10D8_in_san;
sub uni10D8_in_san by uni10D8 gr_in_san;
sub gr_in_san uni10E1' by GR_san;
```

6. Both letters changed and connected.

```
[MF] ligtable GR_UN: GR_SAN =:| GR_un_;
ligtable GR_un_: GR_SAN |=:|> gr_en_san
```

```
[VTP] IN_CONTEXT RIGHT ENUM GLYPH "uni10E1" GLYPH "GR_san_"
END_ENUM END_CONTEXT
SUB GLYPH "uni10E3" WITH GLYPH "GR_un_" GLYPH "gr_en_san" END_SUB
.....
IN_CONTEXT LEFT GLYPH "gr_en_san" END_CONTEXT
SUB GLYPH "uni10E1" WITH GLYPH "GR_san" END_SUB
SUB GLYPH "GR_san_" WITH GLYPH "GR_san_" END_SUB
```

```
[FEA] sub uni10E3' uni10E1 by GR_un_en_san;
sub GR_un_en_san by GR_un_gr_en_san;
sub gr_en_san uni10E1' by GR_san;
sub gr_en_san GR_san' by GR_san_;
```

7. The medial letter changed and kerned.

```
[MF] ligtable GR_KAN: GR_IN =:| GR_kan_;
```

Here is a weak point in METAFONT: After processing the second and third character there is no *simple* means how to return before the first letter and correct kerning (7 right).

```
[VTP] IN_CONTEXT RIGHT GLYPH "uni10D8" END_CONTEXT
...
SUB GLYPH "uni10D9" WITH GLYPH "GR_kan_" GLYPH "gr_en_in" END_SUB
...
FIRST ENUM ... GLYPH "uni10D5" ...
SECOND GLYPH "GR_kan_"
...
i i BY POS END_POS POS ADV -80 DX -80 END_POS
```

```
[FEA] sub uni10D9' uni10D8 by GR_kan_en_in;
...
sub GR_kan_en_in by GR_kan_gr_en_in;
...
@GEOkan_ = [ GR_kan_ ];
@GEOkanA_ = [ ... uni10D5 ...
...
pos @GEOkanA_ @GEOkan_ -80;
```

3.3.8. Lookup order in FEA

According the specification of the feature files [5] the lookup order is determined by the order of their definitions in the file. Their calling order, for example, in a feature block is irrelevant. The order has to be changed by a manual swapping or permutation of the whole text blocks using any text editor. To avoid any misunderstanding it was decided to arrange the order of my lookup definitions:

```
lookup SubstGeorSingleA {
.....
```

```

} SubstGeorSingleA;
lookup SubstGeorInsertA {...} ...
lookup SubstGeorSingleB {...} ...
lookup SubstGeorSingleC {...} ...
lookup SubstGeorInsertC {...} ...
lookup SubstGeorConnC {...} ...
lookup SubstGeorDoubleC {...} ...
and the order of their invocation
feature ss12 { # "Stylistic Set 12"
lookup SubstGeorSingleA; # stage 1: subst one
lookup SubstGeorInsertA; # insert - step 2
lookup SubstGeorSingleB; # stage 2: subst one
lookup SubstGeorSingleC; # stage 3: subst one
lookup SubstGeorInsertC; # insert - step 2
lookup SubstGeorConnC; # subst two - step 1
lookup SubstGeorDoubleC; # subst two - step 2
} ss12;

```

in exactly the same way and thus to “synchronize” both lookup sequences. Having the lookup definitions in one order there is no chance to change this order by trying to call them in any other order.

Resuming my experiences – several conditions must be fulfilled: lookups of different types must be divided in the differently named lookup blocks; the lookups of the same type may be joined together into the common lookup block; and, of course, all the lookups must be arranged in the appropriate order. Putting all the lookups within the font into a single one-level block would be impractical, if not impossible, although it has not been verified.

3.4. Tests of generated fonts

The OpenType tables have been defined, the corresponding VOLT project file created and the VOLT based font by VOLT generated. Also the feature file has been created and the FEA based font generated.

After successful tests of the VOLT based font by VOLT Proofing tool and testing both VOLT and FEA fonts using testing window in FontForge the fonts with OT tables are obtained and ready to be check and tested with X_YL_AT_EX and C_ON_TE_XT and the results could be presented.

The VOLT based font gives the expected results with X_YL_AT_EX (x_el_at_ex):

The VOLT font with **context** prints very similar output:

Also the FEA based font seems to be correct with x_el_at_ex:

სიე ცხედრებზე ნიშნავ

But the FEA font generated by `FontForge` and processed with `context` produces evidently wrong output with many incorrect substitutions:

სიე ცხედრებზე ნიშნავ

It looks like a total nonsense, and may signal a possible incompatibility between `FontForge`, `LuaTeX` and the author's fonts but it could not be said where exactly the bug was.

Let's try the next program package producing OpenType – AFDKO.

3.5. AFDKO

AFDKO (Adobe Font Development Kit for OpenType) is a free program package for OpenType font management. One of the programs, *makeotf*, is able to output only OTF with CFF tables but this fact is not important for us because we are interested mainly in OT features, and description of glyph outlines plays secondary role.

First thing we have met is a small syntactic difference between feature files read by `FontForge` and by *makeotf* (AFDKO).

`FontForge` fails on

```
@GDEF_Ligature = [quotedblleft quotedblbase
  minus dash ];
#@GDEF_Mark = [ ];
@GDEF_Component = [quoteleft comma hyphen ];
table GDEF {
  GlyphClassDef @GDEF_Base,
    @GDEF_Ligature, , @GDEF_Component;
} GDEF;
```

because it does not allow commas in GDEF; while AFDKO corrupts on

```
@GDEF_Ligature = [quotedblleft quotedblbase
  minus dash ];
@GDEF_Mark = [ ];
@GDEF_Component = [quoteleft comma hyphen ];
table GDEF {
  GlyphClassDef @GDEF_Base
    @GDEF_Ligature @GDEF_Mark @GDEF_Component;
} GDEF;
```

because it must have the commas in GDEF, and `@GDEF_Mark = [];` is invalid.

All other definitions (features, lookups, sub and pos rules) are absolutely identical. Some warnings are reported during generating OTF by *makeotf*; however,

the reason for the failure could not be found. Generally, the source inputs for **FontForge** and **AFDKO** are nearly identical, in contrast to the entirely different **VOLT** project regarding syntax and structure.

Reading the appropriate input feature file by **AFDKO** we should satisfactorily generate the **OTF** file. The first extensive tests with **CONTEXT** look like a great success:



All tested substitutions seems to be correct also in **X_{FT}EX**. No mistake has been found in the **GSUB** table:



However, not all positioning rules work properly.

The **GPOS** table produced by **AFDKO** is not correct (not compatible with **X_{FT}EX**) although with **FontForge** we did not observe such problem.

3.5.1. *Final mix*

The last attempt mixes (using **TTX**) the font generated by **AFDKO**'s *makeotf* connected together with the **GPOS** table created by **FontForge**, where both **GSUB** and **GPOS** have been derived from the common source feature file.

X_{FT}EX:



CONTEXT:



Only now no mistakes are seen.

3.6. A short intermediate summary

There are several different source (textual) representations of **OpenType** tables.

1. **VOLT** project (**hwu.vtp**) – the correct font, (**hwuv.ttf**) *only* **TTF** flavoured can be produced.

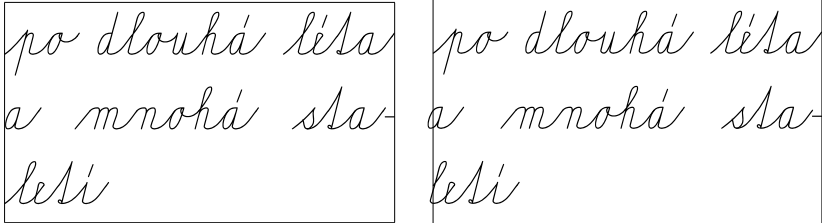
2. First feature file (**hwuf.fea**) – **FontForge** produces **OpenType** with erroneously ordered lookups.

3. Second feature file (**hwuff.fea**) – **FontForge** produces **OpenType** working with **X_{FT}EX**, errors with **CONTEXT** (**GSUB** looks correct).

4. Third feature file (`hwufa.fea`) – *makeotf* (from AFDKO) generates the font (only CFF flavoured) with wrong GSUB; GPOS looks correct.

The 5th OT font (`hwufo.otf`), combined together with TTX from the product of AFDKO (GSUB) and GPOS generated by **FontForge**, works properly with X_YTEX, only small errors are observed in CONTEXT.

The “boundary processing” does not work in the line breaking points, including the points of word hyphenation. The following examples demonstrate the problem – in X_YTEX (left) and in CONTEXT (right):



where the initial (isolated) ‘a’ is not adjusted; ‘a’ in “sta-” does not have the “final stroke”; and the next ‘l’ is without the “initial stroke”.

3.7. Other programs

Alongside with the software tools generating OpenType fonts several other programs have been employed or tried, mostly for the purpose to find errors, verify, compare, convert font data or acquire any relevant information about fonts and their OpenType features. Unfortunately, many of them could not respond to requested questions and do not give any important information.

Predominantly, work has been done on Linux systems, while MS Windows has been used rarely: AFDKO (*makeotf*) and VOLT only to generate VOLT based and FEA based fonts, and VOLT for proofing as well.

3.7.1. Visual proofing tools and displaying binary data in readable form

MS VOLT “Proofing Tool” is very sophisticated and powerful facility. It allows to test the result of complete processing of a given glyph sequence (the glyphs must be denoted by their names according the VOLT project and separated by commas), to check all features separately or step by step detailed behaviour of each lookup, and even to trace the changes glyph by glyph in the string.

FontForge can create and modify PostScript, TrueType, OpenType, SVG, and other fonts; in addition, it comprises other suitable instruments. The “Kerning Metrics Window” allows to check kernings and other features. The results of application of actually selected (activated) features for entered Unicode glyph string can be examined.

In `CONTEXT` the `\showotfcomposition` command provides similar tracing during lookup processing, prints all intermediate results and informs about the features and lookups that have been just applied, step by step until the final result. The “only” a crucial problem is that it is not clear why the activated lookup has not been applied or why is the behaviour of `CONTEXT` and `XYTEX` different when processing my font.

The internal `FontForge` format (SFD) has a readable ASCII representation. The `"Print"` command provides displaying and printing font tables and sample multiscrptal and multilingual texts. Another program from the `FontForge` package, `showttf`, displays a font file tables, and `mensis` allows you to examine and modify some of the tables in a TrueType or OpenType font. But usually an overview of tables and subtables can say nothing about the exact font behaviour and about interaction or interference of features and lookups.

TrueType and OpenType fonts can be converted by the program `ttx` to/from a human-readable XML-based format (TTX). This textual data may be modified using any plain text editor. It was difficult to orientate oneself and it was possible to make only minor changes.

3.7.2. *Validation*

“MS Validator” has been tried only once when the present author’s font did not work properly. A very long output file with complete list of tables, items, features, lookups, etc. in my font was obtained but the only information was that the font is without errors and, of course, nothing about behaviour of the font and the feature execution order.

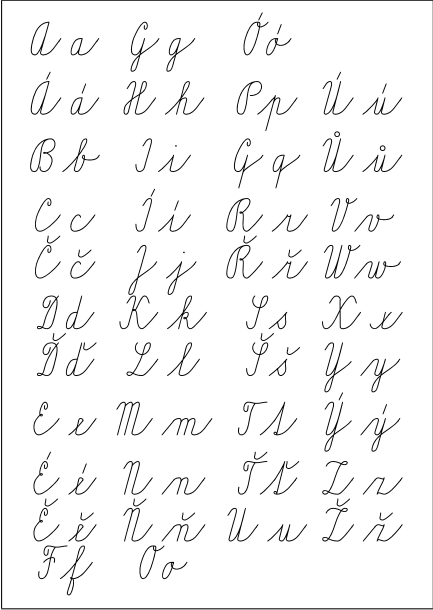
3.7.3. *Comparison*

To compare TTX files is possible, but it is purposeful only if changes are small. Also `FontForge`’s `"Font compare"`, and its command version `sfddiff` afford low benefit if there are significant differences between fonts, e.g., a comparison of a VOLT based font and its FEA version produces vast amount of data, greater than in both fonts, because their internal structures are dissimilar and totally unmatched.

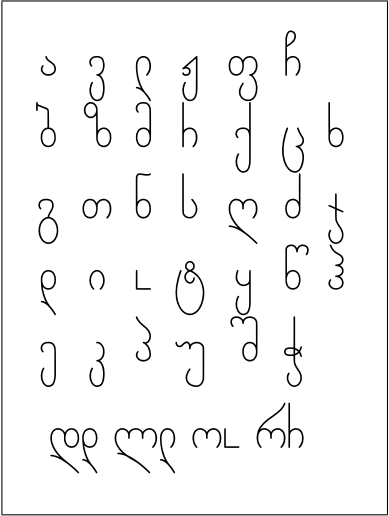
3.7.4. *Conversions*

The programs from the `Font::TTF` package allow to process TrueType/OpenType fonts: `ttf2volt` creates VOLT project or VOLT based font from existing OpenType font file, while `volt2ttf` compiles VOLT source into OT tables in the font.

It was not successful: `volt2ttf` ends with “Can’t use an undefined value as an ARRAY reference at /usr/local/bin/volt2ttf line 574” and for the result of `ttf2volt` (the conversion was executed without errors, only some warnings were reported) VOLT always colapses showing an uninformative message “Compilation failed”.



Czech alphabet.



Georgian alphabet (in the last line: long and short letter variants of d, l, o, r).

With FontForge we can generate a font in other font format; the features of an opened font can be saved into a feature file that can be reread later. However, these files are very similar to input FEA written manually, and – for VOLT based files – are too complicated, less transparent, probably incorrect and unusable.

Therefore such facilities have been found to be rather purely theoretical.

4. Be positive

Two types of font can be produced:

1. TTF flavoured – with VOLT generated from VTP,
2. OTF (CCF/PS) flavoured – with common effort of AFDKO, FontForge and TTX generated from FEA.

The fonts work properly (i.e. corresponding to actually defined substitution a position rules) under $X_{\text{e}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ (`xelatex`) and under $\text{L}^{\text{u}}\text{a}\text{T}_{\text{E}}\text{X}/\text{c}^{\text{o}}\text{n}^{\text{t}}\text{e}^{\text{x}}\text{T}$ – only with some small errors. One could be satisfied despite of many deadlocks during font development and the fact that the font collection has not been finished completely.

The original METAFONT Czech font `slabikar` was created by Olšák [2]. Czech language uses Latin script with extensions, however, local traditions may

be different from other Latin-script languages. The letters in words are always all connected together.

Both Czech and Armenian handwriting are usually slanted and use uppercase and lowercase letters. Modern Georgian script does not distinguish capital and small letters and handwriting is traditionally upright (at least, in the form taught in schools). Not all adjacent letters in words are joined together.

Besides the 'liga' feature the feature names were chosen from the user "stylistic sets":

- 'ss01' for single substitutions to replace letter variants;
- 'ss02' – Czech substitutions;
- 'ss03' – Czech positioning rules;
- 'ss12' – Georgian substitutions;
- 'ss13' – Georgian positionings, e.g. kernings.

For the Czech part to set on all features is obligatory. For Georgian it is possible to select more combinations, only the kerning adjustment is requested in all cases to avoid gaps and letter overlaps.

X_YTEX/X_YL^AT_EX can define the font features (for testing purposed our fonts in many version are not "installed" but they are located in the current directory), for example Georgian without substitutions and without letter connections:

```
\font\hwugn="./hwufo.otf":+liga,  
+ss13" at 28pt
```

ყველა ადამიანი იბადება თავისუფალი და თანისწორი
თავისი ღირსებითა და უფლებებით. მათ მინიჭებული
აქვთ გონება და სინდისი და ერთმანეთის მიმართ უნდა
ექვედნენ ძმობის სულისკეთებით

Georgian with all defined substitutions and letter connections:

```
\font\hwugs="./hwufo.otf":+liga,  
+ss01,+ss12,+ss13" at 28pt
```

ყველა ადამიანი იბადება თავისუფალი და თანისწორი თავისი
ღირსებითა და უფლებებით. მათ მინიჭებული აქვთ გონება
და სინდისი და ერთმანეთის მიმართ უნდა ექვედნენ ძმობის
სულისკეთებით

CONTEXT uses other commands to flip/flip the features:

```
\definefontfeature[cz] [script=DFLT,lang=df1t,  
mode=node,liga=yes,ss02=yes,ss03=yes]  
\font\hwuc = hwufo*cz at 26pt
```

*Všichni lidé se rodí svobodní a sobě rovni
co do důstojnosti a práv. Jsou nadáni
rozumem a svědomím a mají spolu jednat
v duchu bratrství.*

5. Conclusion

It has been possible to generate the fonts with OpenType tables producing the expected results, especially with X_YTEX and CONTEX_T. There are several different representations of OpenType data: the OpenType specification itself, VOLT project source format, feature language and its interpretations in AFDKO and FontForge. Subsequently, the internal binary files produced by various programs should be and (really) are (very often very) different and then also any effective comparison is impossible. Unfortunately, the program tools like AFDKO, FontForge, FontUtils and other have problems either with uniformity and reliability or with compatibility with the TEX based text processors like X_YTEX or CONTEX_T. However, some errors in fonts cannot be excluded even when correct results are produced.

Acknowledgements

I would like to thank all authors of OpenType software, Adam Twardoch for consultations about OT, Hans Hagen and Taco Hoekwater for information about CONTEX_T and its font support.

References

- [1] Donald E. Knuth. *The METAFONTbook*, Volume C of *Computers and Typesetting*, Addison–Wesley, p. 317, 1986.
- [2] Petr Olšák. *Psané písmo ze slabikáře*. Zpravodaj CSTUG 4(7), pp. 191–197, 1997; petr.olsak.net/ftp/olsak/slabikar; bulletin.cstug.cz/pdf/bul974.pdf; Jiří Žáček, Helena Zmatlíková. *Slabikář*, Alter, 1996, 2006 (in Czech).
- [3] Karel Piška. Georgian scripts. *TUGboat*, 19(3), 1998; <http://www.tug.org/TUGboat/Articles/tb19-3/tb60pisk.pdf>.
- [4] Adobe: OpenType. <http://www.adobe.com/type/opentype/>; Microsoft Typography: What is OpenType? <http://www.microsoft.com/typography/WhatIsOpenType.mspx>.
- [5] Microsoft: OpenType specification. <http://www.microsoft.com/typography/otspec/>.
- [6] Microsoft: Visual OpenType Layout Tool (VOLT). <http://www.microsoft.com/typography/VOLT.mspx>.
- [7] Adobe: OpenType Feature File Specification. http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html; http://partners.adobe.com/public/developer/opentype/afdko/topic_feature_file_syntax.html.
- [8] George Williams. Font creation with FontForge. *EuroTEX 2003 Proceedings*, *TUGboat*, 24(3):531–544, 2003; <http://fontforge.sourceforge.net>.
- [9] Jonathan Kew. The X_YTEX typesetting system. <http://scripts.sil.org/XeTeX>; <http://www.ctan.org/tex-archive/info/xetexref/XeTeX-reference.pdf>.
- [10] Martin Hosken. Font-TTF, FontsUtils. <http://search.cpan.org/~mhosken/>; <http://scripts.sil.org/FontUtils>.
- [11] CONTEX_T and LuaTEX. <http://wiki.contextgarden.net>.

*Institute of Physics, Academy of Sciences
Prague, Czech Republic
piska (at) fzu (dot) cz*



www.kostverlorenvaart.nl/context2010

Next to this proceedings we have published in English:

2010

- Petr Rozmahel, Jarko Fidrmuc, Iika Korhonen, Lubor Lacina, Antonin Rusek (eds.): Financial and Economic Crisis: Causes, Consequences and the Future. ISBN 978-80-87106-38-9 (CD-ROM).
- Lubor Lacina, Petr Rozmahel, Antonin Rusek: Financial Crisis: Institutions and Policies. 268 A5 pages. ISBN 978-80-87106-36-5 (softcover). ISBN 978-80-87106-37-2 (CD-ROM).
- Petr Klímek: Applied Statistics for Economics. 120 A5 pages. English-Czech Statistical Glossary is included. Introduction to XLStatistics is also included. ISBN 978-80-87106-32-7 (softcover).
- Pavel Stríž (book of kanjis): Roman Alphabet, Japanese Kana and Kanji Seen Through the Kanji Stroke Order Font and Their Common Drawings in Free TrueType Fonts. 478 A4 pages. ISBN 978-80-87106-31-0.

2009

- Pavel Stríž, Jozef Říha: \TeX , \LaTeX and Friends: Thesis Preparation and Beyond. 326 A4 pages. ISBN 978-80-87106-29-7 (hardcover). A version of next book without Petr Nevřiva's bachelor thesis.
- Pavel Stríž, Jozef Říha, Petr Nevřiva: Using Typographic System \TeX , Its Friends, the \LaTeX Format and Its Packages. 438 A4 pages. ISBN 978-80-87106-28-0 (hardcover).
- Pavel Stríž (nicknamed as book of fonts 3): The Small Book of Easily Available Fonts for Personal, Private and Commercial Use: Book Three. 554 A3 pages. ISBN 978-80-87106-21-1 (hardcover).
- Pavel Stríž (nicknamed as book of fonts 2): The Small Book of Easily Available Fonts for Personal, Private and Commercial Use: Book Two. 448 A4 pages. ISBN 978-80-87106-20-4 (hardcover).
- Pavel Stríž (nicknamed as book of fonts 1): The Small Book of Easily Available Fonts for Personal, Private and Commercial Use: Book One. 436 A4 pages. ISBN 978-80-87106-19-8 (hardcover).

2007

- Pavel Stríž: Mapping and Solving Marketing-informatics Challenges of Forthcoming Knowledge-based Society Efficiently (doctoral thesis series). 276 A5 pages. ISBN 978-80-87106-05-1 (softcover).

Impressions

Many thanks for the nice Tshirt, it is the first time that T_EX Tshirt fitted me so nice and I love the design. In Afrikaans: Baie dankie,

Eva van Deventer, South Africa

Thank you so much for an amazing experience in Brejlov. I have fallen in love with the Czech countryside and, of course, with Prague where I only spent two days. Jaroslav [Hajtmár], it was wonderful to meet you and I will always remember our walk through the countryside together. I have a picture of the Trabant that makes me smile. I am enclosing a few photographs for you all. <http://striz9.fame.utb.cz/texperience/2010/jotky/pavneet-arora>

Pavneet Arora, Canada

A wonderful week!!! Your hospitality was... overwhelming! Thanks.

Steffen Wolfrum, Germany

`\starthankyou`

It was a pleasure to be here

... so nice people

... so wonderful location/environment

... so tasty food

... so careful organization

I hope to meet you here again in three years.

`\stopthankyou`

Mojca Miklavc, Slovenia

I enjoyed the CON_TE_XT meeting and it was lot of fun. I'm really looking forward to the next meeting when we're all be together and T_EX all day and night.

Wolfgang Schuster, Germany

`\dorecurse{1000}{THANKS}`

Hraban Ramm Henning, Switzerland

Thanks for wonderful meeting, for occasion to meet nice people, and for perfect organisation.

Piotr Strzelczyk, Poland

Thank you so much for this conference! I got to discover the Czech Republic as well as the members of the CON_TE_XT community!

Alan Braslau, France

Thanks again for organizing and hosting this excellent conference! I've felt at home and inspired all week in the great company of you & friends. Looking forward to meeting gain! All the best from Amsterdam,

Frans Goddijn, The Netherlands



Mill Brejlov and Frans Goddijn (self-portrait), an excellent photographer!

Zpravodaj Československého sdružení uživatelů T_EXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo: Československé sdružení uživatelů T_EXu
vlastním nákladem jako interní publikaci

Obálka: Antonín Strejc

Ilustrace na obálce: Juraj Horváth

Počet výtisků: 1050

Uzávěrka: 20. 2. 2011

Odpovědný redaktor: Zdeněk Wagner

Redakční rada: Ján Buša, Jiří Demel, Tomáš Hála,
Jaromír Kuben, Michal Růžička, Jiří Rybička,
Petr Sojka, Pavel Stríž, Jan Šustek

Technický redaktor: Tomáš Hála

Tisk a distribuce: KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno,
tel. +420 541 245 548

Adresa: ČSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6

Email: cstug@cstug.cz

Zřízení poštovní aliasy sdružení ČSTUG:

bulletin@cstug.cz, zpravodaj@cstug.cz

korespondence ohledně Zpravodaje sdružení

board@cstug.cz

korespondence členům výboru

cstug@cstug.cz, president@cstug.cz

korespondence předsedovi sdružení

gacstug@cstug.cz

grantová agentura ČSTUGu

secretary@cstug.cz, orders@cstug.cz

korespondence administrativní síle sdružení, objednávky CD a DVD

cstug-members@cstug.cz

korespondence členům sdružení

cstug-faq@cstug.cz

řešené otázky s odpověďmi navrhované k zařazení do dokumentu ČSFAQ

bookorders@cstug.cz

objednávky tištěné T_EXové literatury na dobírku

ftp server sdružení:

<ftp://ftp.cstug.cz>

www server sdružení:

<http://www.cstug.cz>

CONTENTS

Ján Kula, Pavel Stríž: Preface	69
Selected Abstracts from T _E Xperience	74
Abstracts without Papers	78
Arthur Reutenauer: Mobile T _E X: Porting T _E X to the iPad	84
Luigi Scarso: Playing with Flash in CON _T E _X T-mkiv	91
Luigi Scarso: MicroTalk – pdfsplit	102
Ulrik Vieth: Experiences Typesetting OpenType Math with Lua _A T _E X and X _L A _T E _X	116
Taco Hoekwater, Hartmut Henkel: Lua _T E _X 0.60	127
Taco Hoekwater: Lua _T E _X 0.63 Short Reference	134
John Haltiwanger: Subtext: A Proposed Processual Grammar for a Multi-Output Pre-Format	140
Willi Egger: Arranging Pages	147
Libor Sarga: Guide T _E X It: Uneasy Beginnings of Typesetters from the Perspective of Non-Typesetters	157
Jan Přichystal: Typesetting of Tables and Lists and Other New Features in T _E XonWeb	166
Timothy Eyre: Con _T E _X t for 'Zines	170
Hans Hagen: MkIV Hybrid Technology	182
Tomáš Hála: Marking Proof-sheets in Publishing Practice and Its Implementation in the T _E X System	301
Karel Píška: Fonts with Complex OpenType Tables	309