EuroTEX 2005

15th Annual Meeting of the European TEX Users

March 7 - March 11, 2005

Abbaye des Prémontrés

Pont-à-Mousson

France

Monday, March 7

09:00	Welcome	
09:15	MOT01	Javier Bezos Mem. A Multilingual Environment for LATEX with Aleph
09:45	MOT02	Yannis Haralambous, Gábor Bella Omega Becomes a Sign Processor
10:30 – 11:00	Coffee Break	
11:00	MOT03	Joachim Schrod, Chris Rowley, Christine Detig A Taxonomy of Automated Typesetting Systems
11:45	MOT04	David Kastrup Designing an Implementation Language for a TEX Successor
12:30 – 14:00	Lunch	
14:00	MOT05	Jim Hefferon CTAN Plans
14:30	MOT06	Denis Roegel MP2GL: prototyping 3D objects with Metapost
15:00	MOT07	Taco Hoekwater Metapost Developments
15:30 – 16:00	Coffee Break	
16:00	MOT08	Péter Szabó Verbatim Phrases and Listings in LATEX
16:30	MOT09	Stephan Lehmke, Arne Jans, Andre Dierker From RTF to XML to LATEX
17:00	MOT10	Jonathan Fine TEX Forever!
17:30	DANTE e.V. General Meeting GUTenberg General Meeting	
20:30	Diner	
22:00 – 23:00	BoF sessions	

Tuesday, March 8

09:00	TUT01	The TEI/TEX Interface Sebastian Rahtz
09:45	TUT02	Frank Mittelbach, Chris Rowley LATEX3 News
10:30 – 11:00	Coffee Break	

11:00	TUT03	Hans Hagen The 16 Faces of a Dutch Math Journal
11:45	TUT04	Adam Twardoch Typographic Perfection with OpenType?
12:30 – 14:00	Lunch	
14:00	TUT05	Gerd Neugebauer Namespaces for ε_{χ}TEX
14:30	TUT06	Patrick Gundlach contextgarden.net: The ConTEXt Wiki
15:00	TUT07	Thành Hàn Thế Experiences with Micro-Typographic Extensions of pdfTEX in Practice
15:30 – 16:00	Coffee Break	
16:00	TUT08	Johannes Küster NewMath and Unicode
16:30	TUT09	Bogusław Jackowski, Janusz M. Nowacki Latin Modern fonts: how less means more
17:00 – 19:00	TUT10	Panel discussion with Hermann Zapf and Donald Knuth 'With a little help from the wizards'
20:00	Gala Diner	

Wednesday, March 9

8:30	WET01	Thomas Feuerstack ProTEXt, a new TEX-Collection for Beginners
9:00	WET02	Jean-Michel Hufflen Bibliography Styles Easier with MlBibTEX
9:30	WET03	Antoine Lejay La machine à formulaires (The Forms' Machine)
10:00	WET04	Frank-René Schäfer ŞäferTEX: Source Code Esthetics for Automated Typesetters
10:30 – 11:00	Coffee Break	
11:00	WET05	Jérôme Laurens The TEX Wrapper Structure: A Basic TEX Document Model Implemented in iTEXMac
11:30	WET06	Stephan Lehmke Case Study of TEX in Commercial Data Based Publishing: Completely Automatic Typesetting of a Large Product Catalogue

12:00	WET07	David Kastrup The Bigfoot Bundle for Critical Editions
12:30 – 14:00	Lunch	
14:00	Excursion	
19:30	Lunch	
22:00 – 23:00	BoF sessions	

Thursday, March 10

09:00 – 12:30	THT01	Sebastian Rahtz, Hans Hagen XML to PDF, where does TEX fit in
09:00 – 10:30	THT02	Stephan Lehmke TEXPower – Dynamic Presentations with LATEX
10:30 – 11:00	Coffee Break	
11:00 – 12:30	THT03	Gerd Neugebauer, Michael Niedermair $\varepsilon\chi$TEX – Under the Hood
12:30 – 14:00	Lunch	
14:00 – 17:30	THT04	Denis Roegel Metapost
14:00 – 15:30	THT05	Staszek Wawrykiewicz TEXLive 2004 Windows Installer
15:30 – 16:00	Coffee Break	
16:00 – 17:30	THT06	David Kastrup Installing and using Emacs, AUCTEX, RefTEX, preview-latex
19:30	Diner	
22:00 – 23:00	BoF sessions	

Friday, March 11

09:00 – 10:30	FRT01	Hans Hagen ConTEXt
09:00 – 10:30	FRT02	N.N. Advanced LATEX
10:30 – 11:00	Coffee Break	
11:00 – 12:30	FRT01 FRT02	*continuing*
12:30 – 14:00	Lunch	
14:00	Farewell	

Contents

Mem. A multilingual package for LATEX with Aleph

Javier Bezos
Typesetter and consultant
http://perso-wanadoo.es/jbezos/
http://mem-latex.sourceforge.net/
jbezos@users.sourceforge.net

Abstract

Mem provides an experimental environment for multilingual and multiscript typesetting with LATEX in the Aleph typesetting system. Aleph is Unicode-savvy and combines features of Omega and eTEX. With Mem you should be able to typeset Unicode documents mixing several languages and several scripts taking advantage of its built-in OCP mechanism and with a high level interface.

Currently still under study and development, Mem is designed to be capable of following the development of Omega and LATEX3, and I'm publishing it to encourage other people to think about the ideas behind it and to discuss the advantages and disadvantages of several approachs to the involved problems.

The project is now hosted in the public respository SourceForge.net to open its development to other people.

Introduction

Until now, the only way to adapt LATEX for it to become a multilingual system is babel; although another systems like mlp (by Bernard Gaulle) or polyglot (by me) have appeared now and then, in practice only babel is used. It exploits TEX in order to accomplish some tasks which TEX was not intended for, like right to left writing and transliterations, but it's clear that the next step requires features not available in TEX. Further, while one can write documents in several languages, babel is esentially a way to change the main language in monolingual documents.

Long ago, Omega and ε-TEX developement started independently and recently a new project named Aleph, combining features from both systems, has been launched. There are several packages for specific languages taking advantage of the features in Omega (devnag, makor, CJK, etc.) and the package omega provided a few macros to ease its use, now expanded with the name of Antomega by Alexej Kryukov [5], but they don't provide a generic high level interface to add a language and to synchronize it with other languages in a consistent and flexible framework. On the other hand, LATEX3 continues evolving and one of its aims is to have built-in multilingual capabilities.

It is in this context that Mem was born. Actually, it was born several years ago with the name of Lambda and presented in the Fifth Symposium on Multilingual Information Processing (Tokyo, 2001), but for several reasons its development was paused.[1] Its goal is twofold: in the short-term, to provide a real working package for Aleph to become useable with LATEX, taking advantage of features like the OCP mechanism; in the mid-term, to use the experience gained with a real life system in order to develop better multilingual environments with LATEX3 and Omega.

The rest of this paper of devoted to highlight some of the issues and therefore it does not intend to be exhaustive. To get a full picture of the package please refer to the manual [3], which is being written at the same time as the package, because I think the documentation is an integral part in the development process. I've divided the topics in two parts, those related directly to TEX, and those related to the Aleph/Omega extensions, particularly to the OCP mechanism.

The TEX part

Organizing and selecting features Language commands are grouped in *components*, with a few predefined ones—namely, names, date, tools and text. At first sight this resembles babel, but in fact this similitude is only superficial, because you are free to organize and to select components. The limit

[1] There is no paper, but you can find the slides on http://perso.wanadoo.es/jbezos/mlaleph.html. In fact, Mem was born even before, in 1996, with the name of polyglot as I shall explain shortly.

would be a component per macro but this does not seem sensible; for example, left and right guillemets could be a single group. On the other hand, too many components would be unconvenient for the user. I think a sort or component/subcomponent model should be devised (eg, `text.guillemets`), and at the time of this writing I'm working on a system to allow even decisions at macro level like `text.guillemets.\lguillemet`.

This poses the problem to determine which components are active at a certain point of the document. There are, of course, systems like those in CSS and other formatting languages based on description rules for transfomations based on content (for example, with the keywords `inherit` and `ignore`). However, TEX allows programmable rules for transformation based on format and such a model seems very limited (and the term "inherit" can be inappropiate in the context of an object-like environment).[2] Unlike CSS, with its closed set of properties, TEX allows creating new properties and therefore new ways to organize the document layout.

There is a proposal from Frank Mittelbach and Chris Rowley [7] based on nesting levels, with comments about the main issues to be addressed, but since this paper is somewhat abstract regarding the possible solutions it's difficult to determine if that model will be enough for many purposes. In particular, it presumes the structure of the document is a tree, and therefore, as its authors point out, the model has to be extended to provide the necessary support of "special regions" that receive content from other parts of the document.

A basic idea in that paper is that there is a base language for large portions of text as well as embedded languages segments, which are nestable. Although in a limited way, these concepts shown at TUG 1997 related to a clear separation between base and embedded languages were present at that time in my own polyglot package (first released early 1997) whose code I used as the base to develop Lambda and now Mem.

On the other hand, Plaice and Haralambous in [9] and I (in Lambda) proposed independently to follow a model based in context information; the versioning system for Omega described in the former has been worked out and much extended from a theoretical point of view in [11] by Plaice, Haralambous and Rowley, with the introduction of the concept of a *typographical space*. Unfortunately, such a model cannot be carried out in full with TEX and it has not been implemented in Omega, but to me it's clear it should be taken as a guide for Mem, and for that matter for any multilingual environment. At the time of this writing I was studying how to tackle this task and the resulting model will be left for a future paper.

[2] See [2]. An English summary is availaible on `http://mem-latex.sourceforge.net`.

Never again default values! In a well-known article published in the TUGboat ten years ago, Haralambous, Plaice and Braams proclaimed "Never again active characters!" [4]. Now I proclaim the end of another source of problems in the babel package—namely, default values. Actually, default values are mainly associated with active characters, but they are also present in macros. Having default values for a certain language is not a bad thing, but when those values are restored every time the language is selected and they cannot be redefined with the standard LATEX procedures then problems arise.

In Mem, a default value in a language is only a proposal, while the final decision is left to the user, which can change it by means of `\renewcommand`, `\setlenght` and similars. No special syntax is required, like for example `\addto\extrasspanish`. The behaviour of language commands is exactly that of normal commands, except that their values change when the language changes.

A macro is made specific for a certain language with `\DeclareLanguageCommand`, which provides a default definition to be used if the users likes it; if you don't like it, you can redefine it, since the default value is not remembered any more. Outside that language, there could be macros with similar names, but they are not language specific (except if defined for another language, of course).

Furthermore, if a language defines an undefined macro, this is only defined in the context of that language and you not are required to provide a default for *another* language, because I firmly believe loading a language should not change at all the behaviour of another language. In other words, with Mem languages are much like black boxes.

A good example could be the Basque language, which places the figure number before the figure name. For that to be accomplished we must make Basque dependent several internal macros. Considering the number of languages and the fact we cannot know *a priory* which changes will be necessary, the fact languages can (or even must) decide which macros have a default value could lead to an unmanageable situation which could even prevent a proper writing of packages, because we don't know if we need to use `\(re)newcommand` or something else.

The Aleph/Omega part

OTP files The OCP mechanism provides a powerful tool to make a wide range of text transformations which are not possible with preprocessors. Since OCPs perform transformations after expanding macros, we can guarantee all characters, and not only that directly "visible" in the document, are taken into account. One of the main aims of Mem is to develop a high level interface for them, because using the Omega primitives is somewhat awkward. Moreover, since OCPs must be grouped in OCP-lists before actually applying them, the advantages of a high level interface becomes aparent—OCP-lists are hidden to users and language developers and they are built and applied on the fly depending on the language and the context, thus avoiding the danger of a combinatorial explosion [11, p. 107]. For further details on how OCPs works, see the Omega documentation [8] and the very useful case study [10].[3]

A key concept in Mem is that of *process*, a set of OCPs performing a single logical task. Very often, a task cannot be carried out by just one OCP, but in more complex cases a set of interrelated OCPs will be necessary. A very good example of this is the `devnag` package for Omega by Yannis Haralambous, where mapping from Unicode to the target font requires three OCPs. At the time of this writing I'm working on OCPs to handle the Latin/Cyrillic/Greek family of scripts, which is being a lot more involved as one could think at first sight, and very likely a set of three OCPs will be necessary to carry out the single process of mapping from Unicode to the T1, T2*n* and LGR encodings.[4] This is particularly true for Greek with its many possible ways to represent the many possible combinations of letters and accents, which is far from trivial.[5]

[3] Still, the former is very technical and the latter is very basic, and unfortunately an "intermediate" manual explaining the implications of OCPs is not available yet, thus meaning developing OTPs must be done very often by trial and error. The Aleph Task Force and I are considering the possibility to write such a manual.

[4] In addition, it should be investigated if several of the tasks done by these OCPs can be delegated to a virtual font.

[5] And the LGR encoding has some odd assignments, like placing GREEK PSILI AND OXIA at `"5E` (`^`) thus having the catcode of superscript. There is another symbol mapped to the backslash. That would not be important except for a longstanding bug in how OCPs treat catcodes which the Aleph Task Force is trying to fix, because it's a critical one. Since there are very few LGR fonts, and very likely their number will not increase, I'm thinking about removing the support for that encoding and instead to write a virtual file. To add further confusion, the Omega standard font `omlgc` moves the Unicode Greek Extended chars to a non standard placement.

It's important to remember where OCPs are not applied: when writing to a file (e.g., the `aux` file), in `\edef`'s, in arguments of primitives like `\accent`, and in math mode. The latter is a serious limitation, and the Aleph Task Force is working on a solution. This means Mem has done very little in these areas, except redefining `\DeclareMathSymbol` to allow higher values.

Extending OTP syntax: MTP files Perhaps the main limitation of OTP files, containing the source code of OCPs, is that the only letters we can use are those in the ASCII range, while for the rest of the Unicode range we must use numerical values. MTP files have been devised to overcome these limitations so that we can use Unicode names instead of numbers (see figure 1). Currently, they are converted to OCP with a little script named `mtp2ocp`, a preprocessor written in Python.

Another addition to OTPs is that it maps spacial characters to several points in the Private User Area whose catcodes are fixed (as defined by the Mem style file). This way, characters like `\`, `{`, `$`, etc., have the expected behaviour even in verbatim mode.

I hope MTP files could help in the near future to make the task somewhat simpler, so suggestions are most welcome. This way we can have prototypes to experiment with, so that in the future `otp2ocp` itself could be extended with new features if necessary. (One of the reasons I use Python is that it's a great language for prototyping.)

Unicode as input encoding Unicode, unlike many other encodings, clearly separates characters and glyphs. This means that at character level, Unicode can introduce controls to provide further information about these characters, including how they should be rendered. It is expected that this information has to be processed in order to decide which glyph to use. Traditional font formats (TrueType and PostScript) do not have this capability or it is limited.

Unicode, considered as an input encoding, is quite different from other encondings and poses several challenges which must be taken into account if we want to read properly Unicode text. Currently, conversions done by LaTeX packages or Omega OCPs just ignore these controls and instead it is supposed the user must supply them with TeX macros.

For example:[6]

- letters with diacriticals, either composed or decomposed,

[6] For some hints on that, see [13]

```
......................
[LATIN CAPITAL LETTER L WITH STROKE]     => <= @"8A ;
[LATIN SMALL LETTER L WITH STROKE]       => <= @"AA ;
[LATIN CAPITAL LETTER N]{botaccent}<0,>[COMBINING ACUTE ACCENT]
                                         => <= @"8B \(*+1-1);
[LATIN SMALL LETTER N]{botaccent}<0,>[COMBINING ACUTE ACCENT]
                                         => <= @"AB \(*+1-1);
[LATIN CAPITAL LETTER N]{botaccent}<0,>[COMBINING CARON]
                                         => <= @"8C \(*+1-1);
[LATIN SMALL LETTER N]{botaccent}<0,>[COMBINING CARON]
......................
[LATIN SMALL LETTER I WITH MACRON]
        => <= [LATIN SMALL LETTER I][COMBINING MACRON];
[LATIN CAPITAL LETTER I WITH BREVE]
        => <= [LATIN CAPITAL LETTER I][COMBINING BREVE];
.....................
[CENT SIGN]             => "\UseMemTextSymbol{TS1}{162}";
[POUND SIGN]            => "\UseMemTextSymbol{TS1}{163}";
[CURRENCY SIGN]         => "\UseMemTextSymbol{TS1}{164}";
[YEN SIGN]              => "\UseMemTextSymbol{TS1}{165}";
.....................
<acc> [COMBINING GRAVE ACCENT]        => "\UseMemAccent{t}{0}";
<acc> [COMBINING ACUTE ACCENT]        => "\UseMemAccent{t}{1}";
<acc> [COMBINING CIRCUMFLEX ACCENT]   => "\UseMemAccent{t}{2}";
```

Figure 1: Several chunks from MTP files using Unicode names. Currently symbols are hardcoded, not an ideal situation.

- ligatures marked with ZERO WIDTH JOINER,[7]
- hyphens, non breaking hyphens, non breaking spaces, etc.,
- fixed width spaces,
- variation selectors,
- byte order mark.

In order to unify the character encoding used in style files, only utf-8 and explicit Unicode values (eg, `^^^^0376`) are used, but that poses the problem with a non-Unicode document since changing the OCP for the input encoding would mean kerning and ligatures are killed. To overcome this well known TEX limitation, input OCPs use an internal switch mechanim to escape temporarily to utf-8 or utf-16 (see figure 2). The trick is to pass information to the OCP with the character `^^1b`, whose meaning in many character encodings is `ESCAPE`, followed by another character with the operation to be performed. I'm not sure if this mechanims is robust enough, but if it were the idea could in the future serve as a way to pass context information to a certain OCP so that its behaviour may be changed, although of course a built-in mechanism as that proposed by John Plaice *et al.* [11] would be preferable.

[7] The semantics of this character has been extended in Unicode 4.0 and now can be used to mark ligatures [12, p. 389ss]

$$\begin{tikzcd}
\texttt{f\textbackslash unitext\{\^{}\^{}\^{}\^{}0069\}} \arrow[dr] & \\
\texttt{fi} \arrow[r] & \text{ﬁ} \\
\texttt{\textbackslash unitext\{\^{}\^{}\^{}\^{}0066\}i} \arrow[ur] &
\end{tikzcd}$$

Figure 2: Entering a Unicode character with Mem does not break ligatures.

LATEX Internal representation This section is devoted in part to a few ideas which I put forward in the LATEX3 list, which was followed by a very long discussion about a multilingual model (or more exactly, multiscript) for LATEX. These ideas lead to introduce the concept of LICR (LATEX internal character representacion). Actually, LATEX has for a long time had a rigorous concept of a LATEX internal representation but it was only at this stage that it got publicly named as such and its importance realised.[8] The reader can find more on LICR in the second edition of *The LATEX Companion*, by Frank Mittelbach and others [6, section 7.11.2].

What LICR does is essentially to ensure there is only a way to represent a certain character so that

[8] Chris Rowley, "Re(2): [Omega] Three threads", e-mail to the Omega list, 2002/11/04.

different input methods (say, `á` and `\'{a}`) lead to the same representation (in that case `\'a`) and that this representation is able to find a correct glyph somehow.[9] The required funtionality for that to be accomplished is splitted in two well know packages—namely, inputenc and fontenc.

As far as I know, no paper explaining the technical details of the LICR has been published, so I'm going to attempt an operational definition. Before doing that, I think remembering different kinds of TEX expansion process is to the point (I exclude one level expansion as done by `\expandafter`):

- `\def` no expansion.
- `\edef` expands anything except non expandable tokens.
- protected `\edef` expands anything except non expandable tokens *and* protected tokens (even if expandable).
- execution expands anything and performs the actions of primitives.

So, we can say LICR is what we get in a protected expansion.

Unicode provides this kind of "internal representation" but without the normalization of LICR. Let's remember Unicode allows representing characters with diacritics in composed form (eg, ä) or in decomposed form (eg, a¨), and that these forms *may* be normalized to either composed or decomposed forms. There are three possibilities:

- normalizing to composed forms.
- normalizing to decomposed forms.
- not normalizing at all.

Decomposition has, in turn, several types, but we won't discuss them in this paper.

The questions here are: Is it possible the preserve the LICR in Mem?; if so, must be the LICR preserved in Mem? Does it fit in the Unicode model?

In order to answer these questions, we must remember the LICR relies heavily in active characters, which will be replaced in Mem by OCPs. Furthermore, macros are expanded and executed (see above) before OCPs are aplied thus making impossible any attempt to catch things like `\'a`. It seems that an alternative method to inputenc/fontenc must be provided.

Once we have an expanded string, characters are normalized to decomposed characters instead of the composed form favoured by the Web Consortium, for example (it should be noted that in the LICR letters are decomposed). The reasons are

$$\begin{tikzcd}
 & \boxed{\backslash\mathtt{u}\{ę\}} \\
\boxed{ę̆} \arrow[ur] \arrow[r] \arrow[dr] & \boxed{\backslash\mathtt{u}\{\backslash\mathtt{c}\{\mathtt{e}\}\}} \\
 & \boxed{\backslash\mathtt{c}\{ĕ\}}
\end{tikzcd}$$

Figure 3: Several ways to input the same character. With Mem the four are strictly equivalent, because they are converted to Unicode and normalized. With the NFSS, if ę does not exist, then the ˘ is always faked. However, with Mem, if ę does not exists but ĕ does, then ¸ is added to the real composite character.

mainly practical, because the composed form to be selected in some cases depends on the glyphs available. Since normalizing to composed forms would require decomposing, sorting diacriticals and then composing, and font processes would require decomposing again and sorting again to see if there are matching glyphs for the first accent above or the first accent below (or even a combination of both), by using directly the decomposed form we are avoiding a lot of overhead (see figure 3). In fact, the Unicode book says [12, p. 115]:

> In systems that *can* handle nonspacing marks, it may be useful to normalize so as to eliminate precomposed characters. This approach allows such systems to have a homogeneous representation of composed characters and maintain a consistent treatment of such characters.

This dual representation of characters is what is making processes for the Latin/Cyrillic/Greek script so complex, but we have to deal with them if we want a Unicode typesetting engine.

The Latin script has a rich typographical history, which not always can be reduced to the dual system character/glyph. As Jaques André has pointed out, "Glyphs or not, characters or not, types belong to a class that is not recognized as such" [1]. Being a typesetting system, neither Aleph nor Mem can ignore this reality, and therefore we will take into account projects like the Medieval Unicode Font Initiative (MUFI)[10] or the Cassetin Project. However, it doesn't mean a Unicode mechanism will be rejected when available. For example, ligatures can be created with the ZERO WIDTH JOINER. If there

[9] Note the LICR is not necessarily a valid input method, because `\'a` is not always correct in LATEX.

[10] `http://www.hit.uib.no/mufi/`

is a certain method to carry out a certain task in Unicode, it will be emulated.

Diacritical marks The Unicode 4.0 book states [12, p. 184] when discussing spacing modifier letters:

> A number of the spacing forms are covered in the Basic Latin and Latin-1 Supplement blocks. The six common European diacritics that do not have encodings there are added as spacing characters.

In other words, except for these six diacritics (U+02D8-U+02DD), the spacing forms of combining characters are those in the range U+0000-U+00FF. Unfortunately, it happens this is not true, since the spacing caron accent (U+02C7) is not encoded in these blocks. Further, one of these six diacritics encoded separately—namely, the tilde U+02DC—does exist in these blocks (U+007E).

What to do, then? One will be forced to find some kind of hint, and one can do it readily—all characters in the block Spacing Modifier Letters are prefixed with MODIFIER LETTER, except the six spacing clones and CARON (U+02C7). From this, we can infere that the right spacing form for the circumflex accent is not the MODIFIER LETTER variant, but the one in the Basic Latin Block, exactly like the ACUTE ACCENT. No doubt the "small" tilde has been encoded separately because the ASCII tilde has already a special meaning in several OS's.

Still, I think there is a better solution, or rather a better encoding which does not pose this problem. Since the glyphs for diacritics are mainly intended for use with the `\accent` primitive, one can conclude they are, after all, combining characters. The fact we need further processing with TEX does not prevent considering these glyphs conceptually as non-spacing characters—this is just the way TEX works. Since composing diacritical marks are encoded anew in Unicode, we don't need to be concerned with legacy encodings and their inconsistencies.

Conclusions

In this paper I have scratched only the surface of some topics, which deserve by themselves a whole paper. In addition, many others have not been even treated like for example:

- Hyphenation, including patterns for Unicode-like fonts.
- Automatic selection of languages and fonts depending on the current script.
- Since letters are not active any more, one should be allowed to write `\capítulo` or \κεφάλαιο instead of `\chapter`.
- Fonts—monolythic or modular?
- OpenType—must its information be extracted so that it's under our control? (However, using OpenType fonts with TEX is still a failed subject, although there are interesting projects like XeTEX.[11])

Before finishing this paper, I would like to cite Frank Mittelbach in a message posted to the LATEX3 list:

> The fact that we don't agree with some points in it only means that the processes are so complicated that we haven't yet understood them properly and so need to work further on them.

I hope Mem will provide an environment which would help us (including me) to understand better how OCPs work as well the issues a multilingual system poses.

References

[1] André, Jacques: "The Cassetin Project – Towards an Inventory of Ancient Types amd the Related Standardized Encoding", *Proceedings of the Fourteenth EuroTEX Conference*, Brest (France), 2003.

[2] Bezos, Javier: "De XML a PDF, tipografía con TEX", *Proceeding of the IV Jornadas de Bibliotecas Digitales*, Alicante, Spain, 2003 [in Spanish].

[3] Bezos, Javier: "Mem: A multilingual environment for Lamed/Lambda", 2004, `CTAN: macros/latex/exptl/mem/mem.pdf`

[4] Haralambous, Yannis, John Plaice and Johannes Braams: "Never again active characters! Ω-Babel", *TUGboat*, Volume 16 (1995), No. 4.

[5] Kryukov, Alexej: *Typesetting Multilingual documents with Antomega*, 2003, `TeXLive2003: texmf/doc/omega/antomega/antomega.pdf`.

[6] Mittelbach, Frank, and Michel Goossens: *The LATEX Companion,* Addison-Wesley, 2nd ed., 2004.

[7] Mittelbach, Frank, and Chris Rowley: "Language Information in Structured Documents: A Model for Mark-up and Rendering", `http://www.latex-project.org/papers/language-tug97-paper-revised.pdf`.

[8] Plaice, John, and Yannis Haralambous: "Draft documentation for the Ω system", 2000, `TeXLive2003:/texmf/doc/omega/base/doc1-12.ps`.

[11] `http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=XeTeX&_sc=1`

[9] Plaice, John, and Yannis Haralambous: “Supporting multidimensional documents with Omega”, Fifth International Symposium on Multilingual Information Processing, Tokyo, Japan, 2001, `http://omega.enstb.org/papers/dimensions.pdf`.

[10] Plaice, John, and Yannis Haralambous: “Multilingual typesetting with Ω, a Case Study: Arabic”, `TeXLive:/texmf/doc/omega/base/torture.ps`.

[11] Plaice, John, *et al.*: “A multidimensional approach to typesetting”, *TUGboat,* Volume 24 (2003), No. 1.

[12] The Unicode Consortium: *The Unicode Standard, Version 4,* Addison-Wesley, 2003.

[13] The Unicode Consortium: *Unicode in XML and other Markup Languages*, Unicode Technical Report #20, W3C Note 13 June 2003.

Omega Becomes a Sign Processor

Yannis Haralambous
ENST Bretagne
yannis.haralambous@enst-bretagne.fr
http://omega.enstb.org/yannis

Gábor Bella
ENST Bretagne
gabor.bella@enst-bretagne.fr

Characters and Glyphs

The distinction between "characters" and "glyphs" is a rather new issue in computing, although the problem is as old as humanity: our species turns out to be a writing one because, amongst other things, our brain is able to *interpret* images as symbols belonging to a given writing system. Computers deal with text in a more abstract way. When we agree that, in computing, all possible "capital A" letters are represented by the number 65, then we cut short all information on how a given instance of capital letter A is drawn. Modern computing jargon describes this process as "going from glyphs to characters." If a *glyph* is the image of a writing system's atomic unit, a *character* is an interpretation of that image, an interpretation shared by many glyphs drawn by different people in different places at different times. If all these drawings are equivalent in terms of interpretation, we can consider character as an equivalence class of glyphs. To be operational such an equivalence class must be described in a clear and unambiguous way. This is why we define *character* as being a *description of an equivalence class of glyphs* [7, pp. 53–58], [6].

Arabic text provides a typical illustration ground for the concepts of character and glyph. In Arabic alphabet, letters are contextual, in the sense that a given letter will change form according to the presence or absence of other surrounding ones. When we refer to an Arabic letter and represent it graphically, we use the isolated form. We can also refer to it by its description (for example: ARABIC LETTER JEEM) and this can be considered as description of a "character": the equivalence class of shapes this letter can take in millions of Arabic documents. While there may be millions of instances of this letter, according to Arabic grammar they all belong to one of only four forms: isolated ج, initial جـ, medial ـجـ, or final ـج. Hence, we could choose to have not one but four equivalence classes of shapes: ARABIC INITIAL LETTER JEEM, ARABIC MEDIAL LETTER JEEM, and so on. But are these "characters"?

Answering to this question requires a pragmatic approach. What difference will it make if we have one or rather four characters for letter *jeem*? There will indeed be a difference in operations such as searching, indexing, etc. A good question to ask is: "when I'm searching in an Arabic document, am I looking for *specific forms* of letters?" Most of the time, the answer is negative.[1] Form-independent searching will, most of the times, produce better results and this implies that having a single character for all forms is probably a better choice.[2]

Unicode is a *character encoding.* In other words, it contains *descriptions of characters* and tries hard to define characters properly by avoiding dependence on glyphs.[3]

[1] Arabic words are not always visually segmented as English ones—there is, for example, no guarantee that the first letter of a word will always be in initial form: if a word starting with *jeem* is preceded by the definite article *al*, then the *jeem* will end up being in medial form.

[2] Greek is different: *sigma* doesn't "become" final because it "happens" to be at the end of a word. While medial *sigma* can appear anywhere, final *sigma* is used mainly for the endings of particular grammatical forms and in onomatopeias or foreign words. One would hardly ever search for *both* the final and medial form of *sigma* since their rôles are distinct. To illustrate this, when we abreviate a word by a period at a *sigma* then the latter does remain medial despite being the final letter: φιλοσοφία → φιλοσ. Hence it is quite logical to use distinct characters for medial and final *sigma*.

[3] This is not always the case because of Unicode's tenth founding principle, namely convertibility of legacy encodings—and legacy encodings contain all kinds of things. For example, again in the case of Arabic, the main Unicode Arabic table indeed contains only form-independent "characters." But, hidden towards the end of the first Unicode plane, one finds several hundreds of codepoints containing Arabic letters and ligatures in fixed forms, for legacy reasons. Like human history (or Stephen King's movies) Unicode has shadowy places which people try to avoid and even to forget that they exist.

The character vs. glyph issue is far from being solved. In this paper we give an attempt to transcend it by introducing a new concept: the *sign*.

A *sign* is a set $\{c, p_1 = v_1, \ldots , p_n = v_n, g\}$ where c is a character, g a glyph, and p_i an arbitrary number of named properties taking values v_i. Character, glyph, number of properties, their names and their values can be changed at any time, by special syntax in the input file, or by OTPs, or by interaction with fonts.

Using the term "sign" we clearly refer to a Saussurian linguistics tradition whose relevance for nowadays semiotics needs not to be proven. For Saussure [11, p. 99], sign is the basic discrete unit of meaning. It is a dual entity made up of a *signifier* and a *signified*. For instance, when we write or pronounce the word "tree," the visual or auditory image of this word is the *signifier* and the concept of tree is the *signified*. Inspired of this analysis one could attempt to apply the notion of sign to characters and glyphs, by asserting that glyph is signifier, and character is signified. Nevertheless, in semiotics things are not that simple because linguists generally deal with units of meaning rather than with words *per se*, and even less with letters. A letter inside a word is not considered to be a Saussurian sign.

This is why we are bound to warn the reader that our concept of *sign* is inspired from but not identical to Saussurian sign.

In Principio Creavit Knuth TEXum

How does TEX deal with characters and/or glyphs?

First of all, `.tex` files contain characters. When TEX reads a file, it converts the data stream into tokens. A *token* ([10, §289] or [8], which is an exegesis of Knuth's B) is either a "character token" (that is, two numbers: a character code and the character's "category," which provides the specific rôle played by the given character code, for example whether it is a math mode espace character like $, or a comment escape character like %, or a group delimiter like {, and so on), or a "control sequence token."

If we leave aside for a moment the fact that TEX cannot read character codes above 255, one could claim that "character tokens" can still be considered as "characters." What happens next?

Parsing these tokens, TEX builds node lists (horizontal and vertical). A *node* is a typed atomic unit of information of a list. The amount and nature of data contained in nodes depend on their type. A "character node" [10, §134] (or "charnode") is made of two numbers: a font ID and the position of the glyph in the font table. But the latter is not bound to have any relation whatsoever with its character code. Obviously, we can hardly talk about *characters* at this point: we have crossed the bridge to Glyphland.

Another very interesting node type is the "ligature node" [10, §143]. This one contains a font ID, a glyph position in the font table, and a pointer to a linked list of charnodes. This list is in fact the "decomposition" of the ligature. TEX needs it in case it has to "break" the ligature during paragraph building, for example when a word needs to be hyphenated inside a ligature.

Talking about hyphenation, there is a node called "discretionary node" [10, §145]. This node contains two pointers to horizontal lists, as well as an integer. These horizontal lists are what is typeset when we break a word, before and after the line break (in standard cases the first one contains only a hyphen, and the second one is empty). The integer is the number of nodes of the main horizontal list to delete if the word is hyphenated (in other words: how many nodes to replace by the two horizontal lists we mentioned).

As we see, in the three node types described above only glyphs are used—never characters. There seems to be a contradiction with the very nature of hyphenation: after all, words are hyphenated according to rules given by natural language grammars, and these grammars apply to characters, not to glyphs. Indeed, would you hyphenate a word differently if some letters had calligraphic shapes? Certainly not, but for TEX, these letters are glyphs in font tables, and if variant forms exist, then their positions in the font tables are necessarily different from the standard ones. How does TEX deal with this?

There is in TEX a primitive called `\lccode` (and a corresponding `WEB` macro called *lc_code*). Each glyph in a font participating in hyphenation has necessarily a *lc_code* value. These values are usually initialized in the format.

lc_code is in fact a mapping between glyphs and characters. Hyphenation rules are written using patterns, and patterns use characters. When TEX needs to hyphenate words in a paragraph, it first maps glyphs back to characters using *lc_code* [10, §892–899], and then applies hyphenation rules.

This method seems to work, but the user must, at all times, use the appropriate *lc_code* settings for each font.[4]

[4] It is worth mentioning that *lc_code* has a big advantage after all: it allows *simplification* of hyphenation patterns. Indeed, instead of mapping a glyph to the precise character it represents, one can use equivalence classes of characters.

Let us continue our journey through TEX and see what happens in the final stage. There is no surprise: the information contained in charnodes (namely font ID and glyph position) is output to the DVI file [10, §619]. Ligature nodes are treated similarly [10, §652]: only font ID and glyph position of the ligature remains, provided of course that the ligature has survived hyphenation. Discretionary nodes vanish long before output since either hyphenation has occured and the two horizontal lists pointed by the node have found their way into the DVI file, or no hyphenation has occured and the discretionary node falls into oblivion.

When TEX and the DVI file format were developed, this was the best possible approach: DVI files had to be short, and there was no need to insert more information than required to print. And indeed, a printing device doesn't care whether a glyph is a ligature or whether it derives from hyphenation; printing is done in Glyphland, at the very end of the line of document production process. Even PostScript language didn't change that situation, although it made printing devices more clever (clever enough to interpret a very rich prgramming language).

Dixitque Berners Lee: fiat Web et facta est Web

Things changed when DVI and PostScript were not anymore the only targets of TEX document production process. The Web brought the era of electronic documents in various formats such as PDF, XHTML, etc. These documents allow interaction with textual contents: copy-and-paste of text blocks, searching, indexing, etc.

When we search for a word inside a document, or in a collection of documents, do we care about the shape of its letters? Most of the time, the answer is no. Otherwise, it would be quite hard to find a word in a document written in *Zapfino* or *Poetica*, since one has to predict the precise variant form used for every letter, and there are many of them.

In this kind of situation one would like to interact with the document on the character level. But if PDF or XHTML files are produced by TEX, then the information on characters is lost. A very simple example: if 'fi' is represented in a DVI file as glyph 12 of font cmr10, with no reference whatsoever to the character string 'f-i', then how on earth can we search for the word 'film' by entering characters 'f', 'i', 'l', 'm' in our program's search interface?

For example, in Greek, hyphenation does not (or very rarely) depend on accents and breathings, so we can map all letters with diacritics into base letter classes and write patterns using the latter.

There is no natural solution to this problem. Acrobat Distiller tries to give an algorithmic solution by using PostScript font glyph names ([7, pp. 651–653], [1]). The idea is the following: in PostScript type 1 fonts, glyphs have names (namely the names of PostScript subroutines which contain the Type 1 operator glyph's description); when creating a variant glyph of, let us say, letter 'e', designers are requested to use a name like `e.foo` where `foo` is some description of the variant: the first part of the name identifies the Unicode character and the second, the variant; Distiller goes through all glyph names in all fonts used in a document and maps glyphs to Unicode characters according to their names. There is a similar syntax provided for ligatures (that is: glyphs mapped to more than one Unicode character).

TrueType fonts have a table (called `cmap` [7, pp. 703–706]) dedicated to this mapping: we map (single) Unicode characters to (single) glyphs.[5]

These solutions are sub-optimal. There is no way to modify the mapping between characters and glyphs without hampering with the font, and this is not always desirable.

Instead of finding sub-optimal solutions to a problem which is the consequence of information loss in the DVI file, let us attack the origin of this problem. Is it possible to keep character *and* glyph information all the way long, from input file to DVI (and beyond)?

"How now, spirit! whither wander you?" (Enters Omega$_1$)

One of Omega$_1$'s goals was to achieve Unicode compliance. The least one could expect of Omega$_1$ is an affirmative answer to the final question of previous section: Can we obtain Unicode information in a DVI file?

Before answering that question let us see whether Omega$_1$ is actually different from TEX when dealing with characters and glyphs. It isn't: Omega$_1$ can read 16-bit characters (some versions of it can even read UTF-8 representation of Unicode data directly), but once inside Omega$_1$, Unicode characters become "character tokens" and then charnodes, ligature nodes and discretionary nodes

[5] In fact, things are worse than for PostScript Type 1 fonts: while PostScript glyphs have names (and names are usually meaningful and stable vs. font trasformations), TrueType glyphs are accessed by their "glyph index values" which are plain integers. A TrueType font opened and saved by some font editing software can be re-organized, glyph index values can change without further notice, and hence accessing a glyph directly by its index, without going through the `cmap` table, is quite risky.

all the same as in TEX.

How then does Omega$_1$ manage to do Arabic, a writing system where one just *has* to go from characters to glyphs? By using OTPs [9]. An OTP is an internal filter, applied to "character tokens." It can be compared to a pre-processor but has some major advantages: the fact that only tokens are targeted (not comments, for example), and that the catcode of each token is known and that transformations are applied to selected categories only (usually plain text, that is: catcodes 11 and 12). Furthermore, OTPs have the tremendous advantage of being dynamically activated and de-activated by primitives.

Let us analyse the example of Arabic typesetting via Omega$_1$. When Arabic Unicode characters are read, they become "character tokens" (the first part of the token takes the numeric value of the Unicode codepoint, the second part is the catcode, in this case probably 12). No contextual analysis is done yet. It is an OTP that analyses the context of each glyph, and, using a finite-state machine, calculates its form; the result of the transformation by this OTP is one or more new tokens, replacing the previous ones. These tokens correspond to given forms of glyphs. Other OTPs will act upon them and produce esthetic ligatures, and usually the final OTP will map these tokens to font-specific ones, which in turn will become charnodes, and will end up in the DVI file.

The purpose of keeping only the last OTP font-dependent is to improve generality and re-usability of the previous OTPs. But from the moment we perform contextual analysis we have left Unicode data behind and are travelling in a no-man's land between characters and glyphs. In this de Chirico-like surreal place, characters are more-or-less "concrete" and glyphs more-or-less "abstract." Obviously, if the result is satisfying—and this is the case with Omega$_1$'s Arabic typesetting—it is of no importance how we manage to obtain it, whether we go through these or those OTPs and in which ways we transform data.

But the fact is that we do lose character information, just as in TEX. In the DVI file we have beautiful Arabic letters and ligatures ... but there is no way back to the original Unicode characters.

This situation is changing with Omega$_2$ (work in progress). Instead of characters, character tokens and charnodes we are using signs (sign tokens and sign nodes), links and bifurcations. Sign nodes are data structures containing a character, a glyph, and additional key/value pairs, where the value can be simple or complex, involving pointers to other signs, etc. Links are groups of signs which contain alternative sets of glyphs based on a graph: the paragraph builder includes this graph into its own acyclic graph through which it will obtain the optimal paragraph layout as the shortest path from top to bottom.

Before we enter into the details of signs, let us briefly describe another paradigm of character/glyph model implementation: SVG.

The SVG Paradigm

SVG (= Scalable Vector Graphics, [4]) has at leats one very nice property: the way text is managed is quite elegant.

First of all, an SVG document is an XML document, and the only textual data it contains is actual text displayed in the image. In other words: however complex a graphic may be, all graphical elements are described solely by element tags, attributes and, eventually, CDATA blocks. Not a single keyword will ever appear in textual content.

This is not at all the case of LATEX, where we are constantly mixing mark up and contents, as in:

```
Dieser Text ist \textcolor{red}{rot}.
```

where `red` is markup and `rot` is text, and where there is no way of syntactically distinguishing between the two. In SVG, this example would be:

```
<svg:text>
Dieser Text ist
<svg:tspan color="red">rot</svg:tspan>.
</svg:text>
```

where separation between text and markup is clear.

In SVG, as this is the default for XML, text is encoded in Unicode. In other words, text is made of characters only. How then do we obtain glyphs?

As in TEX, SVG uses the notion of "current font," attached to each `text` or `tspan` element, and this informations is inherited by all descendant elements, unless otherwise specified. Fonts can be external, but the model is even more elegant when fonts are internal.

An internal SVG font is an element called `font` containing elements called `glyph`. The latter has an attribute called `unicode`. This attribute contains the one (or more, in case of a ligature) Unicode characters represented by the glyph.

The contents of the glyph element can be arbitrary SVG code (this is quite similar to the concept of TEX's virtual fonts, where a glyph can contain an arbitrary block of DVI instructions). In this way, any SVG image, however complicated, can become a single glyph of a font. To include this glyph in the SVG graphic one only needs to select the font and ask for the same Unicode character sequence as in the `unicode` attribute of the glyph, in a `text` element.

Besides `unicode`, the `glyph` element takes a number of attributes :

- `glyph-name`: in case we want to access a glyph directly (useful when we have more than one variant glyphs representing the same character);
- `d`: the actual outline of the glyph, if we want to keep it simple and not include arbitrary SVG graphical constructions as contents of `glyph`;
- `orientation`: when mixing horizontal and vertical scripts, how is this glyph oriented?
- `arabic-form`: initial, medial, isolated or final?
- `lang`: use this glyph for a number of given language codes only;
- `horiz-adv-x`, `horiz-adv-y`: the advance vector of the glyph when typeset horizontally;
- `vert-adv-x`, `vert-adv-y`: idem, when typeset vertically;
- `vert-origin-x`, `vert-origin-y`: the origin of the glyph when typeset vertically.

What happens if we want a specific glyph, other than the standard one obtained directly going through Unicode character sequences? We can use element `altGlyph` which allows "manual" insertion of glyphs into text, and the PCDATA contents of which is the Unicode character sequence corresponding to the alternate glyph (in this way we get, once again, textual data for indexing, searching, etc.). But `altGlyph` also takes some attributes:

- `xlink:href`: if the font is described as an SVG `font` element, an XLink reference to the corresponding glyph element inside the font—our way of going directly to the glyph, no matter where it is located: server on the Web, file, font;
- `format`: the file format of the font (SVG, OpenType, etc.);
- `glyphRef`: a reference to the glyph if the font format is other than SVG (the specification provides no additional information, we can reasonably assume that this could be the PostScript glyph name in case of CFF OpenType fonts, or the glyph index value in case of TrueType-like fonts, but, as we said already, this is quite risky);
- `x` and `y`: if the alternate glyph is indeed typeset, then these should be the absolute coordinates of its origin;
- all usual SVG attributes (style, color, opacity, conditionality, etc.).

Let us suppose, for example, that we want to write the word "Omega" with a calligraphic 'e' (font *Jolino*) described in the element:

```
<svg:glyph unicode="e" glyph-name="e.joli"
   d="... its path ..."/>
```

We only need to write:

```
<text>
   <tspan font-family="Jolino">
   Om<altGlyph
   xlink:href="#e.joli">e</altGlyph>ga
   </tspan>
</text>
```

We can conclude by saying that SVG jolly well separates textual contents from markup (characters are provided as contents, glyphs as markup), and that `altGlyph` element comes quite close to the goal of our notion of sign: it provides both a character (in fact, one or more characters), a glyph, and some additional properties expressed by attributes. These are not really entirely user-definable as in the case of sign properties, but one could very well introduce additional attributes by using other namespaces.

When the Signs Go Marching In

In the remainder of this paper we will describe the sign concept in more detail and give some examples of applications. We will use the notation $\boxed{\begin{array}{l}\texttt{c=0061 a}\\ \text{g= a, 97}\end{array}}$ for a sign containing character U+0061 LATIN LETTER A, glyph "a" (position 97 in the current font), and no additional properties.

Using this notation, an initial Arabic *jeem* would need a sign $\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{form=1}\\ \text{g= جـ, 18}\end{array}}$. If we would like to typeset this sign in red color, we would add another property: $\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{form=1}\\ \text{color=red}\\ \text{g= جـ, 18}\end{array}}$.

Here is how it happens: Omega$_2$ reads a file containing Unicode character U+062C. Tokenisation produces sign $\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{catcode=12}\\ \text{g=}\varnothing\end{array}}$ (no glyph for the moment). Then we go through the OTPs for contextual analysis and font re-encoding:

$$\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{catcode=12}\\ \text{g=}\varnothing\end{array}} \xrightarrow{1} \boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{catcode=12}\\ \text{form=1}\\ \text{g=}\varnothing\end{array}} \xrightarrow{2} \boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{catcode=12}\\ \text{form=1}\\ \text{g= جـ, 18}\end{array}}$$

The first OTP provides the contextual form value, without affecting the character (or catcode) value. The second OTP adds the glyph information (which would otherwise be added implicitly when reading the font data).

Now what if we wanted the glyph to be typeset in red? All depends if we want (a) only the specific instance of sign to be red, or (b) all *jeem* characters, or (c) all initial *jeem* characters. In the first case one would manually change the `color` property of this sign to value `red`.[6] In the second case one would use an OTP matching all signs conforming to the pattern $\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ *\\ \text{g}=*\end{array}}$ (where asterisks indicate arbitrary values), and would add (or modify the value of) property `color`. In the third case one would, again, use an OTP but this time matching signs conforming to the pattern $\boxed{\begin{array}{l}\texttt{c=062C}\ \text{ج}\\ \text{form=1}\\ *\\ \text{g}=*\end{array}}$. That way only initial form signs will be catched.

One can also imagine OTP patterns based solely on properties. If we wanted to change the color of all red glyphs into green, we would use an OTP as the following:

$$\boxed{\begin{array}{l}\text{c}=*\\ \text{color=red}\\ *\\ \text{g}=*\end{array}} \longrightarrow \boxed{\begin{array}{l}\text{c=(same)}\\ \text{color=green}\\ \text{(same)}\\ \text{g= (same)}\end{array}}$$

Besides catcode, (Arabic) form and color, one can imagine many other properties: horizontal and vertical offset, hyphenation (or hyphenation preference), penalty, glue, bidirectionality level, language, style, word boundary, etc. We will discuss them while describing selected examples of sign applications.

Locked Properties Whenever we define rules, we also need ways to allow exceptions. In our previous color example, let us suppose that there is a given sign which has to remain blue, despite all OTPs which will try to change its color. Properties can be locked: if $\boxed{\begin{array}{l}\text{c=ج}\\ \text{form=1}\\ \text{color=blue}\\ \text{g=ﺟ, 18}\end{array}}$ becomes $\boxed{\begin{array}{l}\text{c=ج}\\ \text{form=1}\\ \text{🔒 color=blue}\\ \text{g=ﺟ, 18}\end{array}}$, then no OTP will ever be able to change this property. Of course, OTPs can lock 🔒 and unlock 🔓 properties, so if that color has to be changed after all, then it can always be unlocked, modified, and locked again . . .

Signs in Auxiliary Files What's the use of having signs and sign properties if all the information is lost when tokens are written into a file? For example, when signs happen to be in the argument of a `\section` command which will be written in a `.toc` file. Instead of losing that information we will write it into that file (which becomes a *sign file*), and have Omega$_2$ read it at the subsequent run and import signs directly.

Sign Documents And since Omega$_2$ reads and writes auxiliary sign files, why not input the main file as a sign document? One could imagine a sign-compliant text editor, a kind of super-Emacs in which one would attach sign information (characters, glyphs, predefined or arbitrary properties) to the TEX code. One can imagine how simple operations like the verbatim environment would become: if we can fix the catcode of an entire text block to 12, then all special characters (braces, backslashes, percents, ampersands, hashes, hats, underlines) lose their semantics and become ordinary text, LATEX only needs to switch to a nice typewriter font and use an `\obeylines` like command and we're done.

Such a text editor is nowadays necessary when we are dealing with OpenType fonts requiring interaction with the user. For example, the `aalt` feature [7, p. 798] allows choosing variant glyphs for a given character. It would be much more user-friendly to use a pop-up menu than writing its glyph index value in the TEX code. That pop-up menu would insert the glyph index as a sign property, and bingo.

Explicit Ligatures

To understand how we deal with ligatures let us recall how TEX uses them in the first place. When building the horizontal list (this part of code is called the chief executive) every charnode is tested for the presence of eventual ligatures (or kerning pairs) in the font's lig/kern program [10, §1035]. If a ligature is detected then a ligature node is created. There is a special mechanism to ensure that the created ligatures is always the longest one (so that we get 'ffl' instead of an 'ff' followed by an 'l').

This ligature node contains a pointer to a horizontal list containing the original charnodes. These will be used in the paragraph builder if hyphenation is necessary.

If we need to hyphenate inside a ligature then it the lignode is first disassembled into the original charnodes [10, §898] and then a discretionary

[6] Why should we insert the color information into the sign as a property, when we can simply use a macro like `\textcolor`? Because OTPs use buffers and control sequence tokens and character tokens of categories other than 11 and 12 will end the buffer and send the buffered text for processing. If the buffer happens to end inside an Arabic word, then there is no way to do proper contextual analysis since the OTP cannot know what will follow in the next buffer. The only way to obtain a sign string sufficiently long to perform efficient contextual analysis, is to avoid control sequence tokens inside Arabic words, and this is easily achieved by storing information in properties.

node is created with the two parts of the ligature as pre-break and post-break lists [10, §914]. This approach allows only one possible hyphenation inside a ligature—as Don says in [10, §904]: "A further complication arises if additional hyphens appear [...] TEX avoids this by simply ignoring the additional hyphens in such weird cases." This can be quite annoying for ligatures of 4 or more letters containing 2 or even 3 potential hyphenation points, and due to the increasing popularity of OpenType fonts we will get more and more of such ligatures in the future.

In TEX a ligature can be inserted *only* by reading the font lig/kern program. It is impossible to instruct TEX to create a ligature node using control sequences or other means, and hence it is equally impossible to do it in Omega$_1$ using OTPs.

Our goal is to do this in Omega$_2$, using signs. We call such a ligature an *explicit* one (in contrast to *implicit* ligatures contained in fonts). Let us take the example of the 'ffl' ligature in the word "affligé." Let us first suppose that there is no hyphenation point in this word:

c=`a`	c=`f`	c=`f`	c=`l`	c=`i`
hyph=0	hyph=0	hyph=0	hyph=0	hyph=0
g= a	g= f	g= f	g= l	g= i

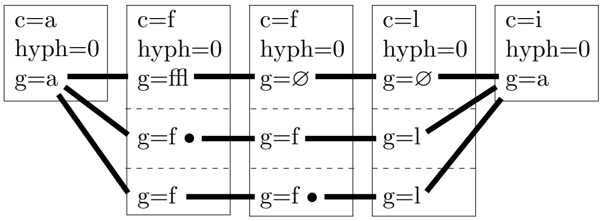
To insert a ligature one would replace it by:

c=`a`	c=`f`	c=`f`	c=`l`	c=`i`
hyph=0	hyph=0	hyph=0	hyph=0	hyph=0
g= a	gdef=f	gdef=f	gdef=l	g= i
	g= ffl	g= ∅	g= ∅	

In this string, character information is left untouched and the ligature glyph is placed in the first sign participating to the ligature (the remaining ones have void glyphs). The `gdef` properties contain the "default glyphs," in case the ligature is broken.

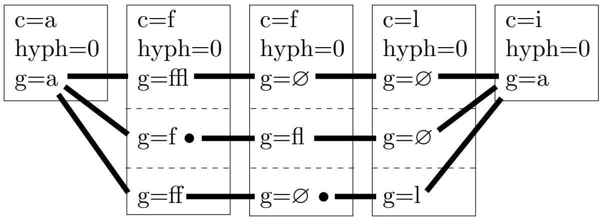
This brings us to a new notion, the one of *link*. The sign string shown in the previous example is, in fact, a doubly linked list. A link is a set of doubly linked signs, in our case those producing the ligature. We say that they participate to the link. The reason for linking these signs is that, at any moment, some OTP may insert additional signs between the ones of the link. We have to be sure that when these signs arrive to the paragraph builder, they will produce a ligature only if they are still consecutive, otherwise we will fall back to the default glyphs.

Things get more complicated if there is a hyphenation point. In this case we must provide all possible combinations of ligatured and non-ligatured glyphs. These combinations form an acyclic graph, very much like the one of TEX's paragraph builder, we call it a set of *bifurcations*. In the figure below, we have illustrated a quite complex case: a ligature 'ffi' surrounded by letters 'a' and 'i' and containing two hyphenation points (after the first and the second 'f' letter), a mission impossible for TEX:



The fat strokes in the figure are the vertices of the graph. These vertices will be examined later for eventual kerning pairs or for other ligatures. The bullet after a glyph indicates that at this location we have a mandatory line break.[7] Notice that all `hyph` properties are now set to 0 since the discretionary hyphenation is handled "manually" by bifurcation.

Here is the same figure, completed with 'ff' and 'fl' ligatures which will only be used in cases the original 'ffl' is broken:



Let us not forget that this graph deals only with glyphs. Characters still form a plain (one-dimensional) string, and macro expansion will use signs in exactly the same way as it currently uses character tokens. The paragraph builder, on the other hand, will include this graph as a subgraph of its network for finding the shortest path. Where we have placed a bullet, the paragraph builder will consider it as a mandatory end-of-line preceded by a hyphen glyph.

Non-Standard Hyphenation

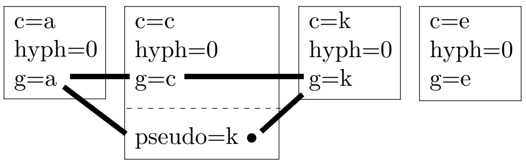
Standard hyphenation corresponds to TEX's `\-`: the first part of the word stays on the upper line and is followed by a hyphen glyph but otherwise unchanged, and the remaining part is placed on the lower line, also unchanged.

But "there are more things in heaven and earth, Horatio." Typical examples of deviant hyphenation are German words containing the string 'ck' (which,

[7] The purpose of this bullet is to postpone until the very last moment the creation of a sign $\boxed{\begin{array}{c}c=\varnothing\\ g=\text{-}\end{array}}$ followed by a line break. The user should be able to decide whether character properties of line break hyphens should be void or U+00AD SOFT HYPHEN, or ant other character.

together with ‘ch’ is a ligature in traditional German typography in the sense that glyphs are brought closer to each other) or Hungarian ‘ssz’ which in some cases is hyphenated ‘sz-sz’ (*ösz-sze*) and in other cases (when the word is composite, like in *kis-szerű*) ‘s-sz’.

Obtaining this using bifurcation is very easy:



The paragraph builder will have to choose between an unbroken glyph string ‘ack’ and a string ‘ak’ followed by a hyphen, a line break, and another ‘k.’ We can insert this information in signs very early, it will remain alive until the paragraph builder. On the character level we keep ‘ack’ so that in text extraction or in conversion to a file format without explicit line breaks (like XHTML) we will always keep the regular ‘ack’, whether or not there has been hyphenation in the DVI file.

There are similar phenomena involving punctuation or diacritics: in Polish, when a word is broken after an explicit hyphen, then we get a hyphen at the end of line, and another hyphen at line begin. In Dutch, ‘oe’ is pronounced ‘ou’ unless there is a diaeresis on the ‘e’; when a word is broken between ‘o’ and ‘ë’, then the diaeresis disappears (since breaking the word at that point makes it clear that the two letters do not form a diphthong). In Greek we have exactly the same phenomenon as in Dutch.

It should be interesting to note that this situation of discrepancy between visual information and text contents is being taken into account by formats like PDF. Indeed, version 1.4 of PDF has introduced the notion of *replacement text* where one can link a character string (← the characters) with any part of the page contents (← the glyphs) [2, p. 872]. The example given is the one of German ‘ck’ hyphenation:

```
(Dru) Tj
/Span
<</ActualText (c) >>
BDC
(k-) Tj
EMC
(ker) '
```

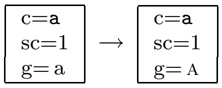
The `ActualText` operator specifies that the string “c” is a “logical replacement” for the contents of the `BDC`/`EMC` block, which contains precisely the string “k-.” As we see, using sign properties to keep this particular information until the DVI file (and beyond) makes sense since PDF is already prepared for handling it, and by using it one can enhance user-interaction with the document.

OpenType Features

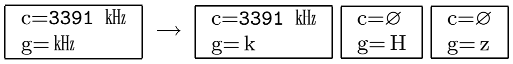
OpenType fonts contain information on various glyph transformations. This works roughly in the following way: the user activates “features,” for each feature the font attempts “lookups” (pattern matching on the glyph string), and for each matched lookup there is a series of glyph positionings or glyph substitutions. In our case, activated features become sign properties (so that they can be entered and modified at any time, independently of macro expansion, and so that they are carried over when tokens are stored, moved or written to files), then at some point, chose by the user, lookups are applied to sign strings, and the effect of positionings and substitutions is again translated into sign properties, before the signs arrive to the paragraph builder.

Both glyph substitution and positioning act on the glyph part of signs only. Let us review briefly the different types of OpenType transformations [7, p. 746–785]:

- *single substitution*: a glyph is replaced by another glyph. For example, a lowercase letter is replaced by a small cap one;

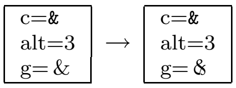


- *multiple substitution*: a glyph is replaced by more than one glyphs. For example, we may want to replace the ideographic square ㎑ by the glyph string “kHz”:

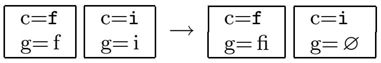


We generate additional signs with empty character parts so that eventual interaction between the glyphs of these signs is possible (for example, they may kern or be involved in some other OpenType transformation).

- *alternate substitution*: one chooses among a certain number of alternate glyphs for a given sign. The user provides the ordinal of the desired variant glyph as a sign property:



- *ligature substitution*: the ordinary ligature. Once again we have to use glyph-less signs:



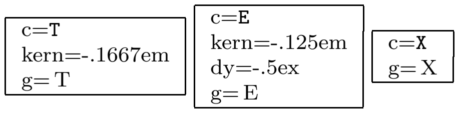
- *contextual substitution*, *chaining contextual substitution*, *reverse chaining contextual substitution*: meta-substitutions where one has a pattern including glyphs before and/or after the ones we are dealing with, with eventual backtrack and lookahead glyph sequences, and sometimes going from end to start;
- *single adjustment*: this is a transformation that adjusts the position of a glyph. In TEX, you can move a glyph horizontally by using `\kern` of `\hskip` commands, and vertically by putting it in a box and `\raise`-ing or `\lower`ing that box. In both cases you lose hyphenation and kerning, and since control sequence tokens are involved, OTP buffers are terminated.

  This is why we introduce two very important sign properties: `dx` and `dy`. They provide horizontal and vertical offsets without going through control sequence tokens. There is no boxing, the advance vector of the glyph does not change, and the effect of moving the glyphs around does not affect surrounding boxes. In other words: even if you raise a glyph using `dy`, this will not affect your baseline—it is rather like if you had used a `MOVEUP` instruction in a virtual font.

  Our favourite example of such a transformation: the TEX logo (one of the first things people learn about TEX, since it is described on page 1 of the TEXbook) becomes a plain sign string without any control sequence inbetween. Here is the standard TEX code, taken from `plain.tex`:

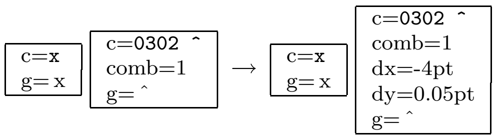
```
\def\TeX{T\kern-.1667em\lower.5ex%
\hbox{E}\kern-.125emX}
```

  and here is the sign string:



  (see below for the `kern` property);
- *pair adjustment* is like single adjustment, but is applied to a pattern of two glyphs. Kerning is the most common case of pair adjustment. Besides `dx` and `dy` we also provide `kern` and `vkern` properties for this. The difference with `dx` and `dy` is that the advance vector of the glyph is modified. To see the difference, here is the TEX logo first with a `kern` property and then with a `dx` property on sign 'E': TEX, TEX;
- *cursive attachment* is a very interesting transformation: we define a mark (that is a point in the glyph's coordinate space) on each side of a glyph, and we declare that proper typesetting in this font is achieved when the right mark of glyph n is identified with left mark of glyph $n+1$. This eliminates the need of kerning pairs (both horizontally and vertically) and is ideal for cursive fonts with connected letters (as we used to write on the blackboard in primary school). We define a property called `cursive`, when it is activated Omega$_2$ will first check that the marks exist in the font, then do the necessary calculations, and finally insert `kern` and `vkern` values to match marks;
- *mark to base attachment*: the same principle as cursive attachment, but this time the goal is to connect a "base" to an "attachment." Usually this will be the glyphs of a Unicode base character and the one of a combining character. The simplest example: a letter and an accent. TEX veterans still remember the headaches caused to Europeans by the `\accent` primitive. Since 1990, thanks to the Cork encoding and its followers, we have been able to avoid using this primitive for many languages. But there are still areas where one cannot predict all possible letter + accent combinations and draw the necessary glyphs. To make things worse, Unicode compliance requires the ability to combine any base letter with any accent, and even *any number* of accents!

  To achieve this, one once again defines marks on strategical positions around the letter (accent scan be placed above, beyond, in the center, etc., these positions correspond to Unicode combining classes) and around the accent. When the glyph of a combining character follows the one of a base character, all we need to do is find the appropriate pair of marks and identify them, using `dx` and `dy` properties. Here is an example:

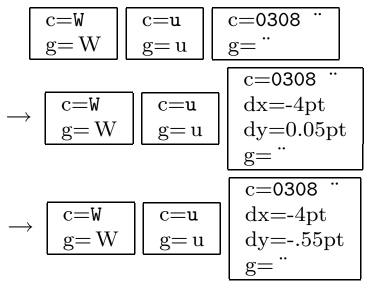


  and the result is 'x̂' (we have deliberately chosen a letter-accent combination which is used in no language we know of, so that there is no chance to find an already designed composite glyph in any font);
- *mark to mark attachment*: instead of attaching the glyph of a combining character to the one of a base character, we attach it to the one of another combining character. The method is strictly the same;

- *mark to ligature attachment*: same principle but things get more complicated since a ligature can have more than one marks in the same combining class and corresponding to individual letters. The idea is to read a ligature of n glyphs followed by n combining glyphs and to place the latter on appropriate locations above (or more generally, around) the former. This is rarely encountered in Latin fonts, but becomes crucial in Arabic fonts with many ligatures (since short vowels and other diacritics are considered as combining characters by Unicode);
- *contextual* and *chaining contextual positioning*: again a meta-transformation where a pattern of glyphs is matched (with eventually a backtrack and a lookahead) and then one or more of the previous positioning transformations are applied. This is crucially missing from TEX.

  A typical example is the acronym S.A.V. (= "Service Après-Vente"), where the 'V' should be brought closer to the period preceding it because the latter is itself preceded by an 'A'. In the case of, for example, S.V.V. (= "Schweizerische Vereinigung für Vegetarismus") kerning between period and second 'V' should better not be applied.

  Another example is German word "Würze," where, in some fonts with a very expansive 'W', the Umlaut has to be brought closer to the letter to avoid overlapping 'W'. In this case we (a) match the pattern of three signs 'Wu¨', (b) place the accent on the 'u', and (c) lower it:



Application of transformations contained in `GPOS` and `GSUB` tables will be considered like activating an OTP, so that the user may insert additional OTPs between those two, or after them.

Doing Better Than OpenType

Positioning of diacritics on Arabic ligatures [3], or of Masoretic diacritics in Biblical Hebrew [5] is a non-trivial task. There are algorithms calculating positions of diacritics using methods such as force-fields, blank area calculation, etc. Until now it is impossible to apply such algorithms without implementing them into the very kernel of TEX.

Using sign OTPs one would first apply contextual analysis, then `GSUB` transformations (and `GPOS` for whatever it is worth) and finally, after the string chain has gone through all OpenType transformations, apply positioning algorithms as external OTPs. At that stage we know exactly which ligatures are used and what the final shape of each word is. The algorithm would obtain the glyphs of ligatures and vowels used—as well as special information such as the presence of keshideh—from sign properties. Having access to the glyph contours of the specific font, it would then reconstruct in memory an envelope of the global graphical image of the word, containing visual centers of individual letters and other relevant information. The result of calculations would be included in `dx` and `dy` properties of vowel signs. After that, processing would continue normally.

In fact, our approach not only uses all resources that OpenType fonts can provide but, contrarily to other systems which rely on OpenType for the final typesetting steps, it allows OTP transformations before `GSUB`, between `GSUB` and `GPOS` and even after `GPOS`. And if we want to use OpenType transformations only partially, we can always lock properties and hence avoid them to be modified by the font.

Meta-information

For TEX, the only way to include metadata (that is: information which is not part of the page description) in a DVI file is through the `\special` primitive. "Specials" are not supposed to interfere with typesetting, but they actually do: if we write

```
A\special{blabla}V
```

there will be no kerning between these two letters. Which means that if we want to change the color of letter 'V' only, we will lose kerning. In Omega$_1$, there is a primitive allowing us to avoid this problem: `\ghost` (which would emulate the behaviour of a glyph related to kerning, without actually typesetting the glyph), but this solution is rather clumsy.

Using signs, we can insert the color information as a property and then include the necessary PostScript code for changing color, long after kerning has been applied (kerning, which, by the way, is also a sign property), or even leave the color property in the DVI file and let (sign-compatible) dvips read that information and write the appropriate PS code.

One could even define sign properties having no effect whatsoever on typesetting. For example, in

Arabic, one could mark the letters *alef* and *lam* of the definite article, or the letter *waw* of the "and" particle, as playing these specific grammatical rôles, so that we can easily distinguish them from letters *alef*+*lam* or *waw* which just happen to be at the beginning of a word. The interest of this lies in the fact that Arabic is not visually separating them from the following word.

Or, again in Arabic, one could imagine a morphological analyser (acting as an external OTP) which would give the letters of the Semitic root of each word a specific sign property. Such letters would be *al*k*i*t*ā*b*u* (the book), k*u*t*u*b*un* (books) *a*kt*u*b*u* (I write), etc. This is the kind of information which would enormously facilitate searching and indexing, but which we would like to avoid representing visually since it would only obstruct reading.

Characters, Sign Properties or Higher Order Markup?

In the previous section we have suggested uses of sign properties which do not affect typesetting. Most often these can also be achieved by characters or by higher order markup.

For example, Ω besides being a popular software project is also a letter of the Greek alphabet *and* the symbol for the SI unit for resistance, named after its inventor Georg Simon Ohm (1789–1854). To distinguish between these two uses of the same symbol, Unicode provides two different characters (`U+03A9` and `U+2126`). Clearly it would be preferable to use one of them should to distinguish between "Omega" and "Ohm," rather than sign properties or higher order markup.

We mentioned the possible use of sign properties for marking the current language. This can seem practical but also has drawbacks: languages are nested, even if their nesting is not always compatible with the logical structure of the document. It would be better to use LaTeX commands for marking languages since these commands will not interfere with micro-typography. Indeed, the author can hardly imagine the need of changing the language of a word in the very middle of it, so that we incur the danger of losing kerning or hyphenation[8]).

Hence, such properties can, at first sight, very well be handled by usual higher level markup.

The best possible use of sign properties is for cases where control sequence tokens would otherwise interfere with the very fragile operations of microtypography and hyphenation.

Glue, Penalty, CJK Languages

Be it fixed or flexible glue, it is now possible, through sign properties, to add it to glyphs, without affecting already existing kerning (which would be added to this glue), ligatures, hyphenation, OTPs that may match the word, etc.

The typical example is letterspacing: how do you i n c r e a s e space between letters[9] while keeping hyphenation of the word, f-ligatures, etc.? Before Omega$_2$, to achieve this, the author was bound to define special font metrics (with tens of thousands of kerning pairs). Now it suffices to add a simple `kern` property to each sign.

Glue for all glyphs is also required in CJK languages where there are no blank spaces between ideographs but where one sometimes needs to shrink or stretch the contents of a line because of a punctuation mark or a closing delimiter which are not allowed to be on line begin, or an opening delimiter which is not allowed on line end. So, even if this is not obvious when reading such text, we do put some glue (with a very small amount of flexibility) between *every* pair of ideographs. In Omega$_1$ this is handled by OTPs, but once such an OTP is used, the ones following it cannot match patterns of ideographs anymore because of the control sequence tokens between them. Once more, it is more natural to systematically add a small amount of glue to each ideograph, using a sign property.

Adding glue to every ideograph is a good thing, but how do we avoid lines starting with punctuation or closing delimiters?

If we can add glue to signs, why not penalties? In that way the space between an ideograph and a punctuation mark or a delimiter will be exactly the same as for all other ideographs, but using an infinite penalty value, line breaking will be prohibited at that point.[10] Here is an example of some ideographs

[8] Although, nowadays, people use more and more language mixtures, like the notorious French *antislash* for "backslash" ... In fact, in French one has anglicisms (French words used with their English meaning, like *librairie* for [code] library, etc.), English words that found their way into French vocabulary (*week-end*, *starlet*, etc.), English words that have been artificially gallisized (*débogage* ← "debugging," *shunter* ← "to shunt", etc.) and many other levels of interaction between the two languages. Hyphenation of these words depends on their level of French-ness, which can vary temporally and geographically.

[9] *Cave canem!* Letterspaced typesetting should be attached to specific semantics and should never be done for justification reasons only, otherwise it is like stealing sheeps.

[10] We don't have that problem in Latin typography because line breaking is allowed only at glue nodes (that is, mostly between words) and inside words using hyphenation—but a punctuation mark has no *lc_code* and hence cannot be matched by a hyphenation pattern.

and the corresponding values of glue and penalty (as used by the author):

c=`9019` 這
glue=0pt
stretch=.025em
g= 這

c=`672C` 本
glue=0pt
stretch=.025em
penalty=10000
g= 本

c=`3002` 。
glue=0pt
stretch=.025em
g= 。

to obtain: 這本。

Glue vs. the "Space Character"

It is well known to us TEX users, that TEX (and thus also DVI) has its own philosophy about how words are separated, namely by glue. The DVI page is like a sea of glue in which glyphs navigate and give the impression of forming words by getting closer to each other. But this is only illusion. In DVI there is no way of distinguishing between, for example, inter-word space and kerning. It is the human eye that deciphers spaces between some letters as being word separators (and the narrower these spaces are, the more difficult is reading). In other markup or typesetting systems, things are different. Unicode defines character U+0020 SPACE as well as a dozen other "whitespace characters." Some of them are extensible and others of fixed width. PostScript uses a mixed approach: when the glyph of the space character has the right width, it is used in strings; when a different width is needed, due to justification, PostScript uses horizontal skips, similar to DVI ones. But in PDF space characters must be present, since people may copy-paste text into other applications: they would be quite surprised to find blank spaces are missing . . .

To be able to distinguish glue produced by interword space from glue entered explicitly, we use a sign for interword glue. This sign has a character part which is one of the Unicode whitespace characters and a blank glyph part. "Blank" is not the same as "void": this sign has indeed a glyph, which can therefore be matched by OpenType lookups, but this glyph has no contour and its advance vector can vary.

Using this approach, not only can OpenType lookups match whitespace glyphs but we can also produce adequate PDF, SVG and XHTML code (for example: in XHTML, interletter kerning should be ignored but interword spaces must be kept in form of Unicode whitespace characters).

Conclusion and Caveats

Work described in this paper is experimental. In other words: what we present here is the latest status of our investigations and experimentations, in the frame of the research project INEDIT of ENST Bretagne. Our goal is to provide a new microtypographical model for typesetting (different from the node-model of TEX) which will be Unicode- and OpenType-compliant, which will provide more control to the user than any Unicode or OpenType-compliant application, and which will produce documents with sufficient information to be converted into any present or future electronic document file format.

There is a discussion list `omega@tug.org` hosted by TUG and dedicated to this project. To subscribe, please visit:

`http://tug.org/mailman/listinfo/omega`

References

[1] Adobe Systems. Unicode and glyph names, 2003.

[2] Adobe Systems. *PDF Reference: Version 1.6*. Addison-Wesley, 5th edition, 2004.

[3] Gábor Bella. An automatic mark positioning system for Arabic and Hebrew scripts. Master's thesis, ENST Bretagne, Octobre 2003.

[4] Jon Ferraiolo, Jun Fujisawa, and Dean Jackson (eds.). *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C, 2003.

[5] Yannis Haralambous. *Tiqwah*, a typesetting system for biblical Hebrew, based on TEX. In *Actes du Quatrième Colloque International* Bible et Informatique*, Amsterdam, 1994*, pages 445–470, 1994.

[6] Yannis Haralambous. Unicode et typographie : un amour impossible. *Document Numérique*, 6(3-4):105–137, 2002.

[7] Yannis Haralambous. *Fontes & codages*. O'Reilly France, 2004.

[8] Yannis Haralambous. Voyage au centre de TEX : composition, paragraphage, césure. *Cahiers GUTenberg*, 44-45:3–53, Nov 2004.

[9] Yannis Haralambous and John Plaice. Methods for processing languages with Ω. In *Proceedings of the International Symposium on Multilingual Information Processing, Tsukuba 1997*, pages 115–128. ETL Japan, 1997.

[10] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[11] Ferdinand de Saussure. *Cours de linguistique générale*. Payot & Rivages, 1916, facsimilé de 1995.

A Taxonomy of Automated Typesetting Systems

Chris Rowley
Faculty of Mathematics and Computing
The Open University, UK

Joachim Schrod
J. Schrod Net & Publication Consultance GmbH
Rödermark, Germany

Christine Detig
J. Schrod Net & Publication Consultance GmbH
Rödermark, Germany

February 21, 2005

Abstract

A long-standing goal is a formal framework and formal description of automated typesetting systems, to capture their functionality and properties in a precise way. As a first step towards that goal, we build a taxonomy for various properties of typesetting systems, both existing and potential.

This taxonomy both establishs a vocabulary and classifies that vocabulary. The vocabulary allows to describe key points and discriminators of automated typesetting systems. Through the classification scheme, we are able to identify and capture the central features of each system, ranging from input and output capabilities to interfaces and models.

The classification is multi-layered; top-level categories are *Formatting Model*, *Document Representations*, *Graphics*, *Font Support*, *Colour & Painting*, and *Style Sheets*. This implies that we do not use a classic tree structure for the taxonomy, leading to a unique position of a typesetting system in the taxonomy tree. Instead, we use a hierarchical cross-classification in which each system can be associated with multiple properties within layers. Selection of properties is guided by the intended use or architecture of the system, not by optional fringe usages and ad hoc extensions.

The taxonomy targets only text typesetting, though for a wide range of natural languages and cultural conventions. Non-classical typesetting, e.g., for screen, interactivity, multimedia, is not covered; neither are associated document processing or management tasks like multi-channel output, indexing, bibliography, version control, editing support, digital rights management, and others.

This taxonomic activity both derives from and enables the analysis of computing systems and thus provides both a vocabulary and a forum for discussing and evaluating current and future developments in the area.

Designing an implementation language for a TEX successor

David Kastrup[*]

February 27, 2005

Abstract

Managing the complexity of TEX's codebase is an arduous task, so arduous that few mortals can hope to manage the underlying complexity. Its original author's computational roots date back to a time where the maturity and expressive power of existing programming languages was such that he chose to employ the assembly language of a fictional processor for the examples in his seminal work "The Art of Computer Programming". In a similar vein, TEX is written in a stripped-down subset of a now-extinct Pascal dialect. Current adaptations of the code base include more or less literal translations into Java (NTS and exTeX), C++ (the Omega-2.0 codebase), mechanically generated C (web2c) and a few others. In practically all currently available cases, the data structures and control flow and overall program structure mimick the original program to a degree that again requires the resourcefulness of a highly skilled programmer to manage its complexity. As a result, almost all of those projects have turned out to be basically single-person projects, and few projects have shown significant progress beyond providing an imitation of TEX.

It is the persuasion of the author that progressing significantly beyond the state of the art as represented by TEX will require the expressiveness and ease of use of a tailor-made implementation and extension language. Even a language as thwarted as Emacs Lisp has, due to its conciseness, rapid prototyping nature, extensibility and custom data types and its coevolution with the Emacs editor itself, enabled progress and add-ons reaching far beyond the original state as conceived by its original authors. This talk tries to answer the question what basic features an implementation platform and language for future typesetting needs should possess.

1 Problems of TEX

Managable problems

- Simplest measures such as `\boxstretch`, `\boxfilstretch`, `\boxshrink` etc are not available.
- Boxes can't reliably be deconstructed (`\special`, single characterse etc. can't be removed, boxes can only be taken apart from the end)
- Variables that TEX employs for decisions are partly unavailable (in some cases because of system-dependent rounding)
- Peculiarities like the loss of the first line's baseline (for `\vtop`) by whatsits, `\splittopskip0pt` and other.

Problems of the macro language

- Only global register pools indexed by number are available. There are no lexically local variables, the grouping structure does not match the macro structure.
- macro arguments get `\catcode`too soon, complex patterns are not easily parseable. Maybe `\lazy\def`would help?
- Implementing regular input languages is hard.

[*] `dak@gnu.org`

Interoperation problems

TEX

- only knows its own font formats, metrics and ligatures.
- does not talk to graphic programs
- can't trigger reformatting of external material.

Algorithmic problems

- TEX is either perfect, or deficient: paragraphs are optimized globally, but the vertical breaks are "local best fit" without feedback to horizontal breaks or future pages.
- TEX has no sane concept for asynchronous user code. `\output` is shielded with the expedient of additional grouping and has no multithreading concept.
- TEX has no possibilities for making use of side-effect free user-defined code. Consequently, user-defined code can't be used in several speculative contexts.

2 Document examples

2.1 Line numbers

Task at hand

If your ultimate goal is to produce a set of files in a different format that can be produced by GhostScript, take a look at the `tightpage` option of the preview package. This will embed the page dimensions into the PostScript code, obliterating the need to use the `-E -i` options to Dvips. You can then produce all image files with a single run of GhostScript from a single PostScript file for all images at once. The `tightpage` option requires setting the `dvips` option as well.
Various options exist that will pass TEX dimensions and other information about the respective shipped out material (including descender size) into the log file, where external applications might make use of it.
The possibility for generating a whole set of graphics with a single run of LATEX, Dvips, and GhostScript increases both speed and robustness of applications. It is to be hoped that applications like LATEX2HTML will be able to make use of this package in future.

Current line number implementations

Implementation with `lineno.sty`:

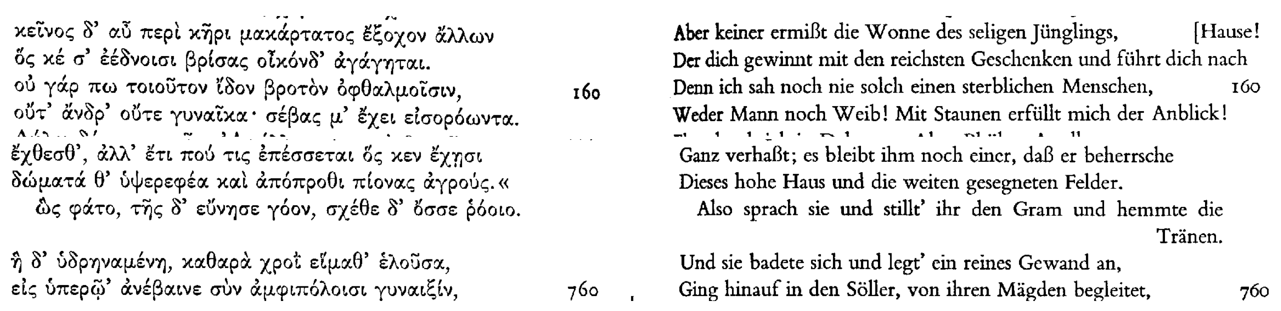
1. Replaces all interline penalties with forced page breaks.
2. This triggers a special output routine placed before the principal output routine.
3. This special routine places the line numbers and reinserts the correct penalties.
4. The normal Output routine is called.
5. A label-like multipass mechanism resets line numbers at the start of the page.

What would be saner for line numbering?

1. For migrating boxes into the main vertical list, a special "context" is defined that assembles a parallel column of 'unfinished' line numbers.
2. The unfinished objects take up constant dimensions and will be translated into glyphs either in the context of the output routine or at shipout time, since then the page start is known.
3. Consequently, a multipass algorithm is not necessary.
4. In the same context `\label`-commands referencing line numbers are expanded.

2.2 More complex Problems

Synchronized texts...



κεῖνος δ᾽ αὖ περὶ κῆρι μακάρτατος ἔξοχον ἄλλων
ὅς κέ σ᾽ ἐέδνοισι βρίσας οἶκόνδ᾽ ἀγάγηται.
οὐ γάρ πω τοιοῦτον ἴδον βροτὸν ὀφθαλμοῖσιν,
οὔτ᾽ ἄνδρ᾽ οὔτε γυναῖκα· σέβας μ᾽ ἔχει εἰσορόωντα.

ἔχθεσθ᾽, ἀλλ᾽ ἔτι πού τις ἐπέσσεται ὅς κεν ἔχῃσι
δώματά θ᾽ ὑψερεφέα καὶ ἀπόπροθι πίονας ἀγρούς.«
ὣς φάτο, τῆς δ᾽ εὔνησε γόον, σχέθε δ᾽ ὄσσε ῥόοιο.

ἣ δ᾽ ὑδρηναμένη, καθαρὰ χροῒ εἵμαθ᾽ ἑλοῦσα,
εἰς ὑπερῷ᾽ ἀνέβαινε σὺν ἀμφιπόλοισι γυναιξίν,

Aber keiner ermißt die Wonne des seligen Jünglings, [Hause!
Der dich gewinnt mit den reichsten Geschenken und führt dich nach
Denn ich sah noch nie solch einen sterblichen Menschen,
Weder Mann noch Weib! Mit Staunen erfüllt mich der Anblick!

Ganz verhaßt; es bleibt ihm noch einer, daß er beherrsche
Dieses hohe Haus und die weiten gesegneten Felder.
Also sprach sie und stillt' ihr den Gram und hemmte die
Tränen.

Und sie badete sich und legt' ein reines Gewand an,
Ging hinauf in den Söller, von ihren Mägden begleitet,

Footnotes in running paragraphs

ösen Neigungen zusammen.[d] Methodisch bedeutsam ist aber[e] wieder die Gewinnung des Endpunktes $^{<}$für die Gegenwart$^{>}$. Dieser[f] muß in einer absoluten und endgültigen Synthese liegen, die eben deshalb nicht aus der natürlichen$^{<}$, ihrem Wesen nach relativistischen$^{>}$ Lebensbewegung [g]stammen oder hervor-

a *In A folgt:* wesentlich **b** *A:* Staatsorganismen,
c–*c* *A:* zukünftige und gegenwärtige
d–*d* *A:* Dass er dabei materiell zu einer sehr konservativen, mittelalterlich ständisch gefärbten und zugleich wieder real-politisch und national gesinnten Staatsauffassung kommt, ist eine Sache für sich. Auch dass die Konstruktion der Entwicklung, die im Grunde immer nur mit einem sehr biologisch getönten Lebensbegriffe arbeitet, kein logisches Fortschrittsprinzip hat, sondern an dessen Stelle sich auf die Vorsehung beruft, ist eine der besonderen Ausführungen des Grundgedankens. Es gibt hier nicht viel mehr als Spielereien mit völlig unzulänglichen historischen Kenntnissen.
e *A:* erst **f** *A:* Er
g–*g* *A:* mit ihrem unaustilglichen Realismus und Relativismus stammen könne

Nested footnotes

$^{<}$dabei$^{>}$ ist, daß alles das immer nur Einzelentwicklungskreise sind[b] und daß der Fortgang zu einer universalen Verknüpfung all dieser Kreise mit dieser Me-

en und Konsequenzen recht interessant, ganz abgesehen von ihrem materiellen Inhalt. Hier über das Problem der Geschichtsphilosophie und des Entwicklungsbegriffes Bd. I S. V und [c]97. Der[c] alles durchdringende Bewegungsbegriff I 5, 49 f., 30, 179, 251. Universalgeschichte und Vorsehung $^{<}$I$^{>}$ 79, 147, 95 f. Zusammenfassung von Smith, Montesquieu und Burke $^{<}$I$^{>}$ 86. Mangel eines archimedischen Punktes $^{<}$für Natur und (offenbarungslose) Geschichte I$^{>}$ 35 f. Die Tendenz des Ganzen [d]III 328: „Den Staat ideenweise (d. h. als Synthese aus Gegensätzen und intuitiv) begreifen heißt ihn für die Gegenwart beseelen, beleben, mit Religion tränken.“[d] [120] $^{<}$Damit ist auch hier der Zusammenhang der Historie und der gegenwärtigen Kultursynthese scharf behauptet.$^{>}$ Die Ablösung Burkes durch De Bonald, Verm. Schriften[e] I 311 ff. Wichtig und interessant ist[f] der „Briefwechsel mit Gentz $^{<}$1800–1829“, Stuttgart 1857. – Außerdem hat mir eine lehrreiche Berliner Dissertation von Georg Strauß über „Die Methode A. Müllers in der Kritik des 19. und 20. Jahrhunderts“[121] vorgelegen$^{>}$.

a–*a* *A:* Romantiker hat dann weiterhin in die Ferne geführt, indische, persische, spanische, französische, englische Geschichte und Geistesentwicklung den Forschern als Gegenstände unterbreitet. Es ist hier nicht möglich, all dem ins einzelne zu folgen und ebenso unmöglich, die mannigfachen Fortwirkungen H. W. Riehl und Gustav Freytag, bis Radowitz und Gierke, Roscher und Knies, Heinrich Leo und Stahl, Boisserée und Schnaase usw. zu schildern, wobei das Hauptinteresse in den jeweiligen Modifikationen läge.

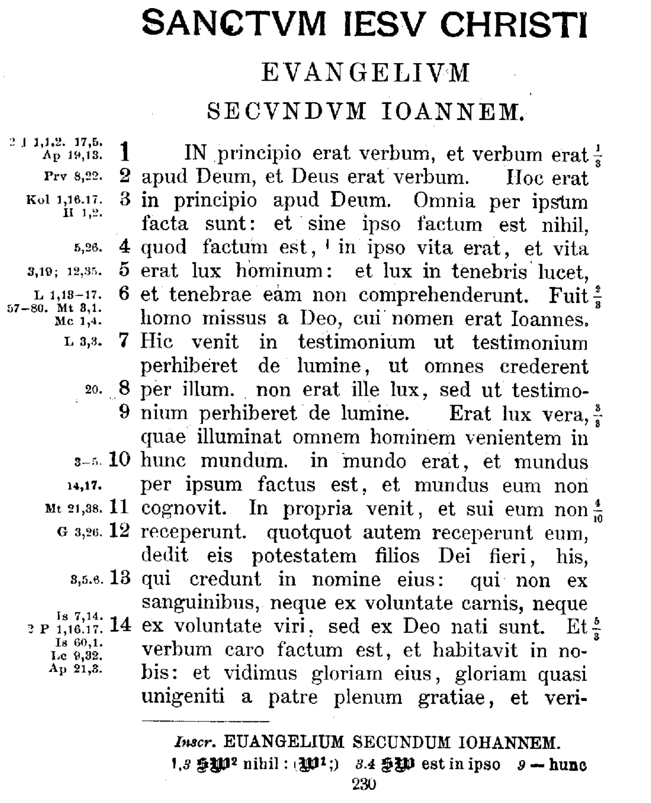
b *A:* sind, **c–*c*** *A:* 97; der

d–*d* *A:* III, 322. Den „Staat ideenweise zu begreifen“ heisst ihn für die Gegenwart „beleben, beseelen, mit Religion tränken.“

e *A:* Schr. **f** *In A folgt:* auch

120 Vgl. Adam Müller: Elemente der Staatskunst, Dritter Theil (1809), S. 238: „Erinnern Sie sich aber, daß es die Grundbestrebung war, den gesammten Staat und alle seine Institute ideenweise zu ergreifen – d. h. ihn zu beleben, zu beseelen, mit Religion zu tränken.“

Tough stuff. . .



SANCTVM IESV CHRISTI

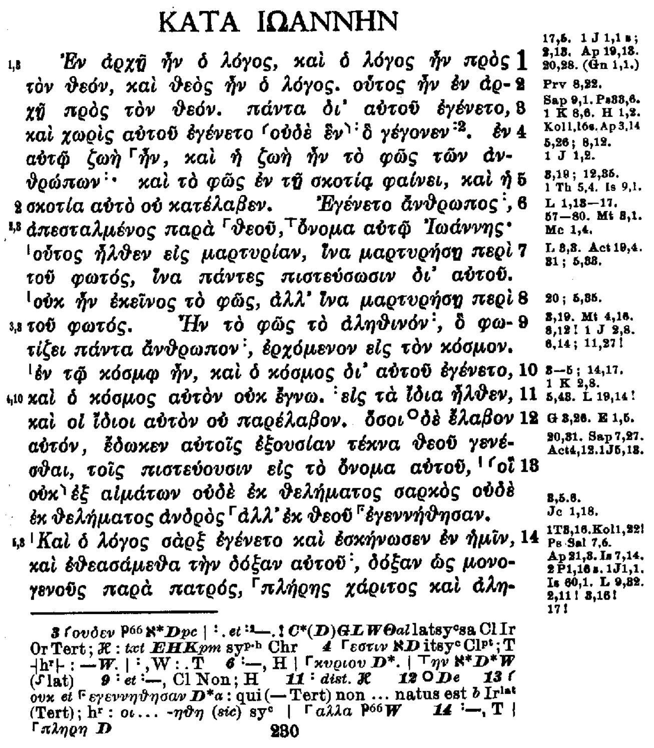
EVANGELIVM

SECVNDVM IOANNEM.

2 J 1,1.2. 17,5. Ap 19,13. 1 IN principio erat verbum, et verbum erat $\frac{1}{3}$
Prv 8,22. 2 apud Deum, et Deus erat verbum. Hoc erat
Kol 1,16.17. H 1,2. 3 in principio apud Deum. Omnia per ipsum
facta sunt: et sine ipso factum est nihil,
5,26. 4 quod factum est, ¹ in ipso vita erat, et vita
3,19; 12,35. 5 erat lux hominum: et lux in tenebris lucet,
L 1,13–17. 57–80. Mt 3,1. Mc 1,4. 6 et tenebrae eam non comprehenderunt. Fuit $\frac{2}{3}$
homo missus a Deo, cui nomen erat Ioannes.
L 3,3. 7 Hic venit in testimonium ut testimonium
perhiberet de lumine, ut omnes crederent
20. 8 per illum. non erat ille lux, sed ut testimo-
9 nium perhiberet de lumine. Erat lux vera, $\frac{3}{3}$
quae illuminat omnem hominem venientem in
3–5. 10 hunc mundum. in mundo erat, et mundus
14,17. per ipsum factus est, et mundus eum non
Mt 21,38. 11 cognovit. In propria venit, et sui eum non $\frac{4}{10}$
G 3,26. 12 receperunt. quotquot autem receperunt eum,
dedit eis potestatem filios Dei fieri, his,
3,5.6. 13 qui credunt in nomine eius: qui non ex
sanguinibus, neque ex voluntate carnis, neque
Is 7,14. 2 P 1,16.17. Is 60,1. Lc 9,32. Ap 21,3. 14 ex voluntate viri, sed ex Deo nati sunt. Et $\frac{5}{3}$
verbum caro factum est, et habitavit in no-
bis: et vidimus gloriam eius, gloriam quasi
unigeniti a patre plenum gratiae, et veri-

Inscr. EUANGELIUM SECUNDUM IOHANNEM.
1,3 𝔖𝔚² nihil : (𝔚¹;) *3.4* 𝔖𝔚 est in ipso *9* — **hunc**

230



ΚΑΤΑ ΙΩΑΝΝΗΝ

1,3 Ἐν ἀρχῇ ἦν ὁ λόγος, καὶ ὁ λόγος ἦν πρὸς 1 17,5. 1 J 1,1 s; 2,13. Ap 19,13. 20,28. (Gn 1,1.)
τὸν θεόν, καὶ θεὸς ἦν ὁ λόγος. οὗτος ἦν ἐν ἀρ- 2 Prv 8,22.
χῇ πρὸς τὸν θεόν. πάντα δι᾽ αὐτοῦ ἐγένετο, 3 Sap 9,1. Ps33,6. 1 K 8,6. H 1,2.
καὶ χωρὶς αὐτοῦ ἐγένετο ⸂οὐδὲ ἕν⸃ ⁚ ὃ γέγονεν ⁚². ἐν 4 Kol1,16s. Ap3,14
αὐτῷ ζωὴ ⸀ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀν- 5,26; 8,12. 1 J 1,2.
θρώπων⁚· καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ 5 3,19; 12,35. 1 Th 5,4. Is 9,1.
2 σκοτία αὐτὸ οὐ κατέλαβεν. Ἐγένετο ἄνθρωπος⁚, 6 L 1,13—17. 57—80. Mt 3,1. Mc 1,4.
3,8 ἀπεσταλμένος παρὰ ⸀θεοῦ, ⸆ὄνομα αὐτῷ Ἰωάννης·
¹οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ 7 L 3,3. Act 19,4. 31; 5,33.
τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι᾽ αὐτοῦ.
¹οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ᾽ ἵνα μαρτυρήσῃ περὶ 8 20; 5,35.
3,3 τοῦ φωτός. Ἦν τὸ φῶς τὸ ἀληθινόν⁚, ὃ φω- 9 3,19. Mt 4,16. 8,12! 1 J 2,8.
τίζει πάντα ἄνθρωπον⁚, ἐρχόμενον εἰς τὸν κόσμον. 6,14; 11,27!
¹ἐν τῷ κόσμῳ ἦν, καὶ ὁ κόσμος δι᾽ αὐτοῦ ἐγένετο, 10 3—5; 14,17. 1 K 2,8.
4,10 καὶ ὁ κόσμος αὐτὸν οὐκ ἔγνω. ⁚εἰς τὰ ἴδια ἦλθεν, 11 5,43. L 19,14!
καὶ οἱ ἴδιοι αὐτὸν οὐ παρέλαβον. ὅσοι °δὲ ἔλαβον 12 G 3,26. E 1,5.
αὐτόν, ἔδωκεν αὐτοῖς ἐξουσίαν τέκνα θεοῦ γενέ- 20,31. Sap 7,27. Act4,12.1J5,13.
σθαι, τοῖς πιστεύουσιν εἰς τὸ ὄνομα αὐτοῦ, ¹ ⸂οἳ 13
οὐκ⸃ ἐξ αἱμάτων οὐδὲ ἐκ θελήματος σαρκὸς οὐδὲ 3,5.6.
ἐκ θελήματος ἀνδρὸς ⸀ἀλλ᾽ ἐκ θεοῦ ⸀ἐγεννήθησαν. Jc 1,18.
5,8 ¹Καὶ ὁ λόγος σὰρξ ἐγένετο καὶ ἐσκήνωσεν ἐν ἡμῖν, 14 1T3,16.Kol1,22! Ps Sal 7,6.
καὶ ἐθεασάμεθα τὴν δόξαν αὐτοῦ⁚, δόξαν ὡς μονο- Ap 21,3. Is 7,14. 2P1,16s. 1J1,1.
γενοῦς παρὰ πατρός, ⸀πλήρης χάριτος καὶ ἀλη- Is 60,1. L 9,32. 2,11! 3,16! 17!

3 ⸂ουδεν 𝔓66 ℵ* *Dpc* | ⁚ . *et* ⁚²—. ! *C**(*D*)*GLWΘal* latsy^csa Cl Ir Or Tert; ℌ : *txt* *EHKpm* sy^p.h Chr *4* ⸀εστιν ℵ*D* itsy^c Cl^pt; T ⊣h^r⊢ : —*W*. | ⁚ ,W : . T *6* ⁚—, H | ⸀κυριου *D**. | ⸆ην ℵ**D***W* (⸉lat) *9* ⁚ *et* ⁚—, Cl Non; H *11* ⁚ *dist.* ℌ *12* ° *De* *13* ⸂ ουκ *et* ⸀ εγεννηθησαν *D***a* : qui (— Tert) non … natus est *b* Ir^lat (Tert); h^r : οι … -ηθη (*sic*) sy^c | ⸀αλλα 𝔓66 *W* *14* ⁚—, T | ⸀πληρη *D*

230

3 Concepts

Contexts

- A context is a programmatic entity with its own control flow and local variables.
- Example: an output context continuously requests material from the main vertical list and insertions. Collections of page matter are then scored (currently this happens using `\brokenpenalty`, `\widowpenalty`, `\clubpenalty`, `\badness` and others).
- The output context thus is coupled with the migration of page material from the vertical list to the current page.
- Other contexts may be coupled with other migrations.
- For example, a color context would have the current color as a local variable for material migrating to the page and into insertions.

Migrations

- Actions get triggered when objects of a class migrate from one list to another.
- Migrations can be penalized.
- When different migrations are possible, the combination with the smallest total penalties survives.
- Line breaking is a special example of penalized breakpoints during the migration of a horizontal into a vertical list.

Objects

- are elements of the various horizontal and vertical lists.
- can belong to different classes.
- classes can be added as well as extended.
- objects can have their own contexts for particular migrations.

Optimization

Global optimization leads to combinatorical explosion of run time. Countermeasures:

1. reduction of interdependencies by separated contexts
2. serialization by tying the optimization to migrations
3. limited backfeed, preferring multiple passes.
4. make do with less than full optimization.

Disadvantages

- higher memory impact since decisions need to remain revertible to some degree.
- higher computational resources because of backtracking
- quite a bit of potential for infinite or almost infinite loops and calculations.
- Programming a full TEX clone on such a platform appears possible, but pointless.
- Decomposition or analysis of several variants can be expensive.

Implementation language

- should offer natural expressivity for lists, TEX-typical strings and token lists.
- should make the required mechanism natively available.
- automatic garbage collection.
- need not be a single layer: instead of TEX's Pascal/TEX-macro layering a more tiered concept like C/Scheme/TEX-core/TEX-Macros would be possible.
- Problematic: Coroutines. Smalltalk? Ada?
- Problematic: I/O (memory for tentative I/O)?
- Combination with low-level languages like C desirable.
- Low-level implementation of fast algorithms on custom data structures should be possible
- Avoidance of unnecessary language features.

CTAN Plans

Jim Hefferon

February 26, 2005

Abstract

CTAN isn't good enough: it needs to change!

It was originally conceived of as an FTP archive, at a time when people using it were typically system managers or advanced users. But today most people have TeX on a personal system, using one of the distributions, and do the maintenance themselves, relying on the web to find information.

In response, CTAN needs to make it easier to find materials, especially using web methods, and to interface with distributions.

DANTE has sponsored meetings between the core maintainers, distribution developers, and others, to discuss how to move forward. This is a progress report; it supplements the article of the same name recently published in several TEX journals.[1]

For the first need, to make CTAN more usable by non-experts, we must make the system more information-rich by including metadata. I will demonstrate the value of metadata – some of the uses to which we can put it to make the life of both CTAN visitors and maintainers easier.

But maintenance of this metadata cannot become just another task, as enthusiasm will inevitably fall off. I will demonstrate some of the efforts to make the metadata easier to maintain. This demo will show a web interface now being tested.

Finally, I will briefly discuss the status of the second need, interfacing with distributions.

[1]See for instance `http://www.tug.org/TUGboat/Articles/tb24-2/tb77heff.pdf`

MP2GL: prototyping 3D objects with Metapost

Denis Roegel

February 25, 2005

Abstract

Metapost was created with 2D graphics in mind, and in spite of various extensions created during the last few years, it doesn't seem well adapted for 3D technical graphics. However, there are cases where simple but realistic 3D graphics are needed, for instance for inclusion in an article, and there are also cases where 3D objects are mere 2D objects with added depth. In such cases, an approach combining metapost with an OpenGL environment proves very interesting and allows for interesting effects. It is also a smooth way to get introduced to OpenGL. MP2GL is our first attempt towards this direction.

MetaPost Developments

Taco Hoekwater,
taco@elvenkind.com

The MetaPost system (by John Hobby) implements a picture-drawing language very much like that of MetaFont except that it outputs Encapsulated PostScript files instead of run-length-encoded bitmaps. MetaPost is a powerful language for producing figures for documents to be printed on PostScript printers, either directly or embedded in TEX documents. It includes facilities for directly integrating TEX text and mathematics with the graphics.

The version number of the MetaPost executable is still well below the 1.0 mark (0.641 is current), but not much has happened in recent years. This situation is far from satisfactory, especially since a fairly large number of bugs are known to exist at this date, but John Hobby simply could not find the time to solve these bugs, let alone handle feature requests.

Resulting from a renewed community interest in MetaPost, last summer a small group of people have made a proposal to Hobby for the creation of a special development group that would take care of the development of MetaPost from then on. Luckily, he agreed, on the condition that he will only allow tested code to be inserted into the MetaPost distribution. Among the currently active group are the following people:

- Karl Berry
- Giuseppe Bilotta
- Hans Hagen
- Taco Hoekwater
- Bogusłav Jackowski

Karl Berry has created a homepage on the TUG server for MetaPost

- `http://www.tug.org/metapost`

He also created a mailinglist for discussions and questions. Details can be found at

- `http://www.tug.org/mailman/listinfo/metapost`

Taco Hoekwater has set up a project at Sarovar that hosts a source repository as well as a bug / feature request tracker

- `http://www.sarovar.org/projects/metapost`

The MetaPost manuals (mpman and mpgraph) have recently been released under a BSD-ish license. Dylan Thurston at Debian converted the sources to LATEX, and in the future they will become a standard part of the distribution.

As of today, the known errors in the documentation have been removed, and a number of bugs have already been fixed in the repository. More bugs will be fixed in the near future, and the group hopes that a new bugfix release will be available around EuroTEX 2005.

Verbatim phrases and listings in LATEX

Péter Szabó ⟨`pts@fazekas.hu`⟩ *

Budapest University of Technology and Ecomomics,
Department of Analysis,
Műegyetem rakpart 3–9.,
Budapest, Hungary H-1111

2004-11-11

Abstract

The examplep package written by the author recently provides sophisticated features for typesetting verbatim source code listings, including the display of the source code and its compiled LATEX or METAPOST output side-by-side, with automatic width detection and enabled page breaks (in the source), without the need for specifying the source twice. Special care is taken so section, page and footnote numbers do not interfere with the main document. For typesetting short verbatim phrases, a replacement for the `\verb` command is also provided in the package, which can be used inside tables and moving arguments such as footnotes and section titles. The listings package is used for syntax highlighting.

The article reviews the design decisions made during the package development and also presents some interesting implementation internals. examplep is compared to standard LATEX packages such as listings, ltxdoc, sverb and moreverb. The new codep package and its accomanying Perl script, which provide a convenient interface to the examplep package for authors of manuals, is also presented. With codep it is possible to generate the source code, the LATEX or METAPOST output and the compilable example file onto the CD from a single source embedded into the appropriate place of the `.tex` document file.

1 Terminology

verbatim text A visually distinguishable textual part of the document (usually typeset with a monospaced, or typewriter font) that is allowed to contain the full ASCII character set. Verbatim text is often used to typeset parts of program source files, including TEX source. Verbatim text must be marked up (i.e. surrounded) in the source of the LATEX document, so backslash and other control characters are typeset verbatim instead of being interpreted as commands or special LATEX characters.

inline verbatim A verbatim text passage inside a paragraph or table cell.

display verbatim is a vertical verbatim text block between paragraphs.

side-by-side display When typesetting program source in a display verbatim, it is often desirable to show the output of the program as well. This is especially useful when teaching scripting languages, so the reader can see the command and its effect side-by-side in a quick glance. For TEX or METAPOST sources and EPS and PDF source files it is also useful to see the source and the typeset result side-by-side.

Source and Sample Side-by-side displays can be divided to *Source* and *Sample*, the latter being program output or typeset material.

CD-files Files accompanying a book, usually on a CD or DVD shipped with the book, or avalable for download on the home page of the book. These files usually contain some of the display verbatim material in the book, so readers do not have to retype them.

*Thanks to Ferenc Wettl for the fruitful long discussion about the line syntax of the `code` environment, and also for reviewing the article.

2 Special characters in the LATEX source

The special meaning of the input characters in the source file must be disabled in verbatim mode – except for the character(s) that delimit the end of the verbatim text. The following characters have to be dealt with:

ASCII symbols The ligatures have to be disabled, especially `?‘` → ¿ etc. The safest way is to make both characters of such a ligature active, and defining `\def?{\relax\string?\relax}` and `\def‘{\relax\string‘ \relax}`. examplep issues similar definitions covering all such ligatures in the OT1- and T1-encoded CM fonts.

ASCII letters Most verbatim fonts don't contain the "fi" or similar ligatures, so examplep takes no care to disable them.

special TEX source symbols To disable the special meaning of the symbols `\ { } $ % ^ & _`, examplep redefines their `\catcode` for display verbatim. However, catcode changes are not always appriopriate for inline verbatim, so examplep provides \÷ (see below) which doesn't change catcodes at all.

other ASCII punctuation Some of these characters (`[ ] ; ’ , . / ~ ! @ * ( ) + | : " < > ? ‘ - =`) may be active, for example, `~` is ÷ ÷, `‘` is a Babel shorthand (e.g. in the Hungarian language module, magyar.ldf); `"` is a Babel shorthand (e.g in the German language module); `?`, `!`, `:` and `;` are activated (e.g. by the French language module), and other characters may be activated, too. So examplep sets the catcode of all characters in the range 33 . . . 126 to other (12). This includes all ASCII punctuation, letter and digit characters.

double carets For example, the letter "J" can be input as its ASCII code in hex, prefixed by double carets: `^^4a`. However, in verbatim mode we want the 4 characters, not the letter J. There is no problem in inline verbatim mode, because `^` loses its special meaning once its catcode is changed. However, when the verbatim material is written to a file or to the terminal, TEX may change "unprintable" characters to escapes prefixed by `^^`. These changes must be reverted when reading the file back. See Subsection 7.2 for more.

high characters Input characters in the range 128 . . . 255 are usually activated by the inputenc package, and they know how to typeset themselves – so examplep leaves the catcode of such characters intact. However, in some modes, examplep has already changed all catcodes to 12 and 10 (with `\meaning`), so it has to change the catcodes of such high codes back to 13 (active).

3 Features of examplep

These are the most important, unique features:

- layout of side-by-side display may depend on maximum Source width
- automatic hyphenation of inline verbatim. The text is divided into words and punctuation symbols (based on catcodes). For words, the normal TEX hyphenation patterns apply, and it is allowed to break the line on both sides of a punctuation symbol.
- customizable isolation of page, section etc. numbers in the Sample and the host document with the `PexaMiniPage` environment
- besides the outer level, inline verbatim (when properly escaped inside `\Q` or \÷) works safely inside macro arguments, section titles, footnotes, table cells and index entries
- generated CD-files with automatic page and chapter number
- writing verbatim data to CD-files with a Perl script; exact, binary reproduction of verbatim text is guaranteed
- ability to write different material to Source, Sample and CD-files

Table 1: Contexts and features of inline verbatim commands

	outer	argument	tablular	elsewhere	escaped
`\verb`	+	−[1]	+	−[1]	no
`\PVerbOpt`	+	+[2]	+	−	no
÷	+	+[2]	−	+[2]	no
\÷	+	+	+	+	yes
`\Q`	+	+	+	+	yes

[1]sometimes displays the proper error message
[2]inner mode only (spaces compressed or lost, `%` is comment etc.)

- the accents `\H` and `\.` work as expected with monospaced fonts in the OT1 encoding. By default, `\texttt{\H o}` produces ő in OT1 encoding, because such typewriter fonts (such as *cmtt10*) have those accents replaced by ASCII symbols. examplep solves the problem by getting the accent from the *cmr* font family.

Some other features:

- side-by-side display of the Source and the Sample
- between-word hypenation of inline verbatim
- customizable left and right indentation of display verbatim
- specifying inline verbatim with nested braces (`\PVerb`, `\Q`) or terminating character (`\PVerb`, ÷, \÷)
- automatic line breaks with hyphenation in display verbatim
- the discretionary hyphen (`\hyphenchar`) of verbatim text is different from the one in normal text
- line numbering in display verbatim
- writing to temporary files only if needed
- reading back contents of any file in display verbatim
- inline verbatim with a single character (÷) and its escaped version (\÷ and `\Q`)
- automatic `\indent`/`\noindent`, based on empty line above `\begin{...}`
- ISO Latin accented input character support in all modes (also present in `\verb`)
- support for syntax highlighting with the listings package [1]
- emits a tab as eight spaces in normal mode, but tabs are supported properly with `ttlistings=yes` and `ttlistings=showtabs`
- simple side-by-side display emulation without temporary files, using `srcstyle=leftboth` or `srcstyle=leftleft`

3.1 Escaped mode of inline verbatim locations

For compatibility reasons, examplep doesn't change the original `\verb` and `\verb*` commands in any way, but defines its own commands: `\PVerb`, `\PVerbH`, `\PVerbInner`, `\PVerbOpt`, `\Q`, ÷ and \÷. Most of the `\PVerb`... commands are historical. For new documents, only the use of `\PVerbOpt`, `\Q`, ÷ and \÷ is recommended. Some of these commands have to be activated with package load options:
`\usepackage[Q=yes,div=yes,bsdiv=yes]{examplep}`. The reason why ÷ was introduced is that it is a high Latin-1 (and Latin-2) character available on the Hungarian keyboard, which is usually not used in LaTeX documents (in fact, `$\div$` is used instead). Inline verbatim sources with such a character are compact, and they can contain all ASCII symbols.

examplep supports inline verbatim text at the outer level, inside macro arguments, in table cells and elsewhere (in section titles, in footnotes and in index entries), see also in Table 1. The reason why some of these cases are treated differently is that catcode changes must be timed correctly so that the proper catcodes are active by the time TEX reads the verbatim text from the input file for the first time. (Please note that section titles and index entries are also written to and read back from auxilary files.)

This is quite hard to accomplish in several cases (because TEX's mouth gathers macro arguments at high speed, a way before TEX's stomach could change the catcodes), so examplep provides the commands ÷ and `\Q`, which do not change catcodes at all, so they work everywhere. Each special (say, not alphanumeric) character of the source text of these commands must be prefixed by a backslash, so TEX's eyes will see it as a controls sequence token. The backslashes are retained when the construct is written to auxilary files, but they get removed upon typesetting. Letters, when prefixed by a backslash, get special meaning, for example `\V` denotes a visible space. For example, The construct `\÷\\\}÷` is seen by TEX's eyes as \÷$_{13}$\\$_{13}$\}$_{13}$÷$_{13}$, and its gets typeset as `\}` (by running the command `\÷`). Please note that the construct is properly nested, because all braces are inside control sequence names. The same result (`\}`) can be achieved with `\Q{\\\}}`. Both constructs are safe, because they can be freely moved to anywhere in the source file. However, for compatibility reasons, `\Q` is recommended, because its execution doesn't rely on the current catcode of the terminating ÷ of `\÷`. A more complicated example: `‘‘\Q{\\\V X\ }’’` gets typeset as "`\␣X` ". In escaped mode, `\V` dentoes a visible, unbreakable space, `\S` and `\␣` denote default space (affected by the `pverb-space=` option), `\B` allows a line break there with a discreationary hyphen (affected by the `pverb-linebreak=` option), and `\n` flushes left and starts a new line.

The `\PVerb` macros can detect whether they have been invoked from within a macro argument. If so, they do not insist on catcode changes, but they emit all the tokens that has been seen by TEX's eyes. (Spaces are already compressed now, and everything after `%` is ignored etc., so this is not purely verbatim anymore.) However, this works only if the macro argument is properly nested with respect to braces, and it is delimited by braces (not a terminator character).

Please note that there might be problems with verbatim material in index entries processed by makeindex if characters `"`, `@`, `!` and `|` are not quoted properly with `"`. This is a generic makeindex issue. The quoting must be applied even inside verbatim material.

3.2 Horizontal alignment of the Source lines

The `verbatim` environment of standard LATEX reads the whole verbatim text into a macro argument, thus limiting the length of the verbatim material to the available main memory. This is enough for about 3400 80-character lines. The verbatim, moreverb, listings and examplep packages parse the input line-by-line, so there is no such limit. However, with examplep, additional memory is required for aligned mode, which limits the maximum number of lines to about 2200 (32 pages) when the average line width is 80 characters. Please note that the maximums mentioned here may be lower if more packages are loaded; in another situatian the maximum number of lines was 375. As a reference, a plain `\halign` with all lines having `(9999)\hfil\cr` only could accomodate about 5000 lines when no packages were loaded. The maximum memory can be increased by increasing the `extra_mem_bot` variable in texmf.cnf or in the environment.

There are two display modes used for display verbatim: paragraph and aligned (see the `\pexa@show@pars` and `\pexa@show@halign` macros, respectively, in the source). Aligned mode is used when multiple columns (such as line numbers and text) have to be aligned horizontally. Aligned mode uses the TEX `\halign` primitive to do the alignment, and this primitive reads the whole construct into memory before typesetting it (in order to be able to calculate the column widths). The other one, paragraph mode, is used when horizontal alignment is not needed and side-by-side display is not used. In paragraph mode, each line is typeset as a seperate paragraph, so the length of the verbatim text is only limited by the available disk space to hold the resulting DVI file. examplep chooses the mode automatically: for side-by-side display it always chooses aligned mode (so it can measure the width of the Source before typesetting it), otherwise, if the `srcstyle=` makes it possible, it chooses paragraph mode, otherwise it chooses aligned mode. See Figure 1 for details about Source styles.

Please note that the Source styles `leftboth` and `leftbothnumcol` display each line twice, as Source and as Sample, too. This is different from regular side-by-side display, because lines of the Source and Sample here are forcibly aligned, and this solution doesn't use a temporary file. The source style `leftleft` is similar, but it lefts the author specify different Source and Sample for the same line (they should be separated by `&` in the source).

```
  srcstyle=left PAF
 9srcstyle=leftnumhang PAF
  9srcstyle=leftnum PAF
   9srcstyle=leftnumcol AF when the last page number has 2 digits
                                 srcstyle=center PAF
                                                          srcstyle=right PAF
  srcstyle=paralign          PF           with         source-par-align=justjust
  srcstyle=leftboth | srcstyle=leftboth A
   9srcstyle=leftbothnumcol | srcstyle=leftbothnumcol A
  srcstyle=leftleft | anything A
```

P: works in paragraph mode
A: works in aligned mode
F: works when Source is read back from file

Figure 1: The effect of the `srcstyle=` option

```
\chapter{My chapter}\label{c}
Welcome\footnote{to our isolated
minipage environment}!
\newpage
\section{My section}\label{s}
The chapter begins on page
\pageref{c}.\par
\begin{equation}a^2+b^2=c^2
\end{equation}
\par Indented.
```

I My chapter

Welcome[1]!

[1] to our isolated minipage environment

1

1 My section

The chapter begins on page 1.

$$a^2 + b^2 = c^2 \qquad (1)$$

Indented.

2

Figure 2: Display verbatim isolation with the `PexaMinipage` environment

3.3 Display verbatim isolation

The `PexaMinipage` environment is provided, which is similar to the built-in `minipage` environment, but provides better isolation of the Sample from the container document, because it saves and restores section, page, equiation (etc.) numbers and also marks (section titles in page headers). Labels (for `\label`, `\ref` etc.) are not isolated, because many packages use them in a non-standard way. See Figure 2 for an example.

The environment also cancels vertical skips (including `\belowdisplayskip`) at the bottom of its contents. For space conservation, `\abovedisplayskip` above the very first displayed equation is also canceled. To vaid this, put `\everydisplay{}` before the formula. The environment starts with `\noindent`, but subsequent paragraphs are indented.

3.4 Feature comparison

Although there are several LaTeX packages providing display and/or inline verbatim environments for LaTeX, examplep has some important unique features not found in other packages (see the beginning of this section for details). The author has tried the following packages before deciding to write examplep:

verbatim Although the `verbatim` environment is built-in into LaTeX, its most important limitation is that it eats up TeX memory when typesetting very long verbatim material (of several hundred or thousand lines). The verbatim package fixes this, and provides the `\verbatiminput` command (similar to the `\PexaShowSource` command of examplep) and the `comment` environment (similar to `PIgnore` in examplep).

moreverb This package extends the verbatim package with additional features: proper handling of tabulators (also accessible from examplep with the listings interface), line numbering (also available in examplep with much more customization), verbatim surrounded by a frame (this is not available in examplep, but it works with listings, with page breaks allowed), the `verbatimwrite` environment writes its contents to a file (similar to the `WFile` environment in examplep).

sverb It provides display verbatim with tabulators and long environments, and it can read and write text from files. It also has a side-by-side environment (`demo`) with fancy frames. The verbfwr package (part of the examplep distribution) was derived from parts of this package.

syntax This package is written by the author of sverb. It provides generic and customizable inline verbatim support and it also has powerful features to typeset BNF-like grammars and syntax diagrams. It is documented that no attempt is made to make the constructs work inside macro arguments or section titles.

alltt This standard LATEX package defines the `alltt` environment in which the characters `\ { }` retain their original meaning, so it is possible to do some manual formatting in the verbatim text.

fancyvrb [5] This is extremely configurable verbatim package provides inline verbatim even in footnotes, display verbatim even with side-by-side, line numbers on any side, all kinds of francy frames even with page breaks, text formatting and writing and reading from files; and very long diplay verbatim text. Options can be specified any time within the argument of the `\fvset` command. The original `verbatim` environment is not modified, but a new one, `Verbatim` is defined. Setting the background color is not possible.

This package is not actively developed. Version 2.7 (dated 2000/03/21) is part of teTEX. Oddly enough, the newest version on CTAN is 2.6, which a file timestamp in 2004, but it dated 1998/07/17).

fvrb-ex This package is part of the fancyvrb distribution and uses the fancyvrb package. It provides a side-by-side display verbatim environment (`SideBySideExample`). A page break is not allowed in the Source after the Sample. The `xrightmargin` option has to be specified manually (e.g. `xrightmargin=3cm`. The first two characters of each line in the environment are ignored.

ltxdoc The most important features of examplep inspired by the LATEX documentation package are display verbatim line numbering with `srcstyle=leftnum` and inline verbatim started with ÷. The ltxdoc package typesets everything between two `|` characters as inline verbatim. This is not supported by examplep to avoid making an ASCII character active.

listings [1] The most important layout elements of this sophisticated, highly customizable, actively developed package missing from examplep are: background color, frames (with page breaks allowed), syntax highlighting and proper tabulator support. Except for the background color and frames, these features can be used from examplep with its interface to listings, see in Subsection 4.1. Use `\lstset{columns=fullflexible,language=C,backgroundcolor=\color{red},frame=trBL}` to try these features. listings also provides inline verbatim mode (with syntax highlighting), in the character-delimited argument of the command `\lstinline`; unfortunately, spaces at line breaks (with option `breaklines`) are rendered in an inconsistent way, and line breaks do not work well with background color.

listings has an important weak point: it cannot typeset ISO Latin accented characters with the inputenc package; the way described in the manual doesn't work as expected: it puts the accented characters to the wrong place in the line. This problem, however, is solved when listings is invoked from examplep. See more about listings in Subsection 4.1.

4 Customization

The operation of examplep can be customized with options as ⟨key⟩=⟨value⟩ pairs. Global options, which affect all subsequent commands within the current block, can be specified as package load options (`\usepackage[...]{examplep}`) or as argument of the `\PexaDefaults` command. LATEX doesn't allow complicated option values (such as values containing some expandable macros, for example after `linenumberformat=`) to be specified in the `\usepackage` line – use `\PexaDefaults` in such cases. Many commands and environment accept local options, which affect only that construct.

All options have default values, which are indicated below right after the option name. If the option has a fixed set of possible values, all of them are mentioned, and they are prefixed by `=`. The defaults have been chosen so the `PSource` environment matches the builtin `verbatim` environment as closely as possible. Note that the original environment is not overridden before the `verbatimenv=yes` is specified.

`Q=unchanged` To enable `\Q`, use `=yes` instead of the default.

`abreak=unchanged` To enable `\abreak`, use `=yes` instead of the default.

`addvspace-bottom={\vskip\z@skip\addvspace}` Specifies the command to add vertical space below display verbantim. The default works fine, but e.g. packages maintaining the baseline grid might want to change it.

`addvspace-top=\addvspace` Command to add vertical space above display verbantim. The default works fine, but for example, packages maintaining the baseline grid might want to change it.

`allowbreak=yes` Use `=no` to disable page breaks in the Source of display verbatim.

`allowshrink=yes` The default will shrink the Sample horizontally if the Source is too wide. Use `=no` to disable this. Use `=force` to enable shrinking of Source, and with `srcstyle=leftleft` or `srcstyle=leftboth`, also enable shrinking of the Source if it is narrow.

`baseline-grid=no` Use `=yes` to adjust the height of the Sample to be an integer multiply of `\baselineskip` (with `yalign=u` and `yalign=v`).

`boxstyle=p` Controls how the Sample is boxed. Capital letters are not allowed for side-by-side display. By default (`=p`), a `PexaMinipage` environment is put inside a `\vtop`. Use `=h` to put a `\hbox` only, `=v` to put a `\vtop` only, `=V` to put a `\vbox` only, `=m` to put a `minipage` environment inside a `\vtop`, `=M` to put a `minipage` environment inside a `\vbox`, `=P` to put a `PexaMinipage` environment inside a `\vbox`, `=G` to add `\begingroup` and `\endgroup` only. The default is recommended for most cases, because the `\vtop` provides proper alignment with the Source, and the `PexaMinipage` environment provides isolation (of page and section numbers etc.) from the main document.

`bsdiv=unchanged` To enable \÷, use `=yes` instead of the default.

`div=unchanged` To enable ÷, use `=yes` instead of the default.

`firstlinenum=1` Specifies the number of the first line of a numbered Source listing.
Useful with `srcstyle=leftnumcol` and `srcstyle=leftnumhang`.

`linenumberformat={{...}}` Commands to display a line number and the separator in a numbered Source listing. See the default value in `examplep.sty`.

`linenumbersep={}` Commands to display the separator in a numbered Source listing.

`listings=no` Use `=yes` to display each Source line with the listings package. Specify options (to be executed in `\lstset`), separated by commas in the argument. Read more about the options in the documentation of the listings package. For example, `listings=yes` and `listings={}` uses listings with default options (no syntax highlighting), and `listings={language=C,showtabs}` enables syntax highlighting for C language and enables visible tabulators. See Subsection 4.1 for more information.

`listings-verbatimfont=pexavf` Set the font to be used in the Source when `listings=` is active.
See `source-verbatimfont=` for the possible values.

`mp-equation-reset=yes` Use `=no` to make the main document and the Samples inside `PexaMinipage` share the same equation counter.

`mp-varioref-reset=no` Use `=yes` to make the internal counter `vrcnt` of the varioref package to be reset for each Sample inside `PexaMinipage`. This option doesn't affect the final output, and varioref expects this counter not to be reset, so it is not recommended to change the default.

`noligs=some` By default, only those ligatures are disabled whose second character is one of `` ` ' , - < > ``. Use `=kernel` to get the same effect, but using the LaTeX built-in `\@noligs`. Use `=most` to get all ligatures with either the first or the second character having code between 32 and 127 and catcode 12. Please note that ligatures in inline verbatim mode are disabled anyway, because `\allowbreak` is inserted between characters of catcode 12, depending on the value of `pverb-linebreak=`.

`pexaminipage-setuphook={}` Extra commands to run when starting the `PexaMiniPage` environment, just after the environment has finished its own initialization.

`pverb-hash=full` Use `=half` to make `\PVerb` convert `##` to `#`. The command `\PVerbH` is the same as `\PVerb` but forces `=half`. This is required when `\PVerb` or ÷ is used inside a macro argument. For example, `\textit{÷#÷}` yields the error message *Illegal parameter number in definition of \reserved@a*, but `\textit{\PVerbH{##}}` works fine. The error message is a general LaTeX kernel limitation, for example `\textit{\@gobble{#}}` doesn't work either.

`pverb-hyphenchar=hyphen` By default, the minus character (ASCII code 45) is used to for automatic word hyphenation in inline verbatim. Use `=char'30` to have character with code 24; see also Subsection 7.6. Use `=none` to disable word hyphenation (by setting `\hyphenchar` to -1). Use `=unchanged` to get the hyphenchar from the font (the *cmtt* and `ectt` fonts have word hyphenation disabled). Please note that hyphenation around symbols is affected the `pverb-linebreak=` option, not this one.

`pverb-leftbreakmin=2` Specifies the minimum number of characters in inline verbatim after which it is allowed to break the line (with `pverb-linebreak=`). Values allowed are 0, 1 and 2, but 0 usually doesn't make sense.

`pverb-linebreak=char` By default, `\PexaAllowBreak` is inserted around symbols in inline verbatim, so a line break (with a discreationary hyphen affected by `pverb-linebreakchar=`) is allowed there. Use `=yes` to insert `\allowbreak` instead, which allows a line break without discretionaries. Use `=no` to disable line breaks around symbols in inline verbatim. The option `pverb-hyphenchar=` affects intra-word hyphenation in inline verbatim, not this one.

`pverb-linebreakchar={$\lnot$}` Specifies the discreationary hyphen to be used in `\PexaAllowBreak`. See also `pverb-linebreak=`.

`pverb-space=invbreak` By default, spaces in inline verbatim are invisible, variable width (as allowed by the font, see also `pverb-stretchshrink=`) and breakable (i.e. a space can be replaced by a line break if necessary). Use `=invdisc` to get an invisible, variable width space which becomes visible (␣) when it is broken at the end of the line. Use `=invfixbreak` to get an invisible, fixed width and breakable space. Use `=invnobreak` to get an visible, variable width and unbreakable space. Use `=visnobreak` to get a visible, fixed width and unbreakable space. Use `=visbreak` to get a visible, fixed width space with line breaks allowed on both sides. Use `=invbreakleft` to get a visible, variable width space with infinite stretchablility if the line is broken there (this may have strange effect on other line breaks in the paragraph, so please try to avoid it). The built-in `\verb*` command uses `source-space=visnobreak`.

`pverb-stretchshrink=yes` By default, spaces in inline verbatim are forced to be stretchable and shrinkable (by `\quad`/9). Use `=no` to disable stretchability and shrinkability. Use `=unchanged` to keep the settings in the font. Note that `\fontdimen`3 and `\fontdimen`4 are changed by this option, and the changes are local to inline verbatim mode.

`pverb-verbatimfont=pexavf` Set the font to be used in inline verbatim mode. See `source-verbatimfont=` for the possible values.

`samplewidth=.5\PexaWidth` Specifies the maximum width of the Sample in side-by-side display as a TEX dimension. The actual Sample can become actually narrower (see `allowshrink=`. The dimensions `\hsize`, `\linewidth` and `\PexaWidth` can be used. (Our LaTeX book used `samplewidth=.45\PexaWidth`.) `\leftskip` and `\rightskip` do not affect this option. `\hsize` can be used, which is the total width available (including the extra margins added by surrounding list environments), `\linewidth` is `\hsize` widtout the extra margins produced by lists, and `\PexaWidth` is the total width of the Source, the separator (see `vrule=`) and the Sample.

`source-par-align=left` Specifies the alignment of Source lines when `srcstyle=paralign` is active. Use `=left` (default), `=right` or `=center` to specified flush-left, flush-right or centered alignment, respectively. Use `=justify` to have the last line flush-left and the previous line justified (please note that each Source line is mapped to a single paragraph, so the paragraph will have more than 1 line only if the source line is too long). Use `=justjust` to have all lines justified. Use `=unchanged` to keep the alignment of the enclosing block.

`source-sepwidth=\tabcolsep` Specifies the horizontal distance between the Source and the Sample. See also `vrule=`.

`source-space=invfixbreak` Specifies how to typeset spaces of the Source. See `pverb-space=` for the possible values. The built-in `verbatim*` environment uses `source-space=visnobreak`.

`source-verbatimfont=pexavf` Sets the font to be used for the Source when `listings=` is not active (see also `listings-verbatimfont=` for `listings=`). Give `=ttfamily` to use `\ttfamily`, `=pexavf` to use `\pexa@@verbatimfont` (which defaults to `\verbatim@font`), `=latexvf` to use `\verbatim@font` (which defaults to `\normalfont\ttfamily`), `=unchanged` to keep the current font, or `=normalfont` to use `\normalfont`.

`srcstyle=left` Specifies the horizontal alignment of the Source lines. See more in Subsection 3.2 and Figure 1.

`ttlistings=` (no default) Shorthand of `listings-verbatimfont=ttfamily,listings=`.

`url=unchanged` To enable `\url`, use `=yes` instead of the default. The `\url` will be defined as `\def\url{\PVerbOpt{}}`. This has the disadvantage that inside `\textit` etc. it cannot typeset URLs having a single `#` (see `pverb-hash=` for more), but the url package has the same limitation. A quick fix: use `\itshape` instead of `\textit` etc.

`usewidth=skipwidth` Specifies which horizontal part of the main text should be used in a display verbatim. By default, left and right margins introduced by list environments (such as `itemize`) and `\leftskip` and `\rightskip` are respected. Use `=linewidth` to ignore `\leftskip` and `\rightskip` but respect list environments. Use `=hsize` to use the whole width of main text. Note that this option affects the calculation of `\PexaWidth`.

`vextrabotdepth=\z@` Dimension to add to the depth of the display verbatim with `yalign=v`. The default works fine, but for example, packages maintaining the baseline grid might want to change it for each instance.

`vextravskip=\z@` Amount of vertical space to be added above display verbantim with `yalign=v`. The default works fine, but for example, packages maintaining the baseline grid might want to change it for each instance.

`vsmallht=1pt` Specifies Sample height threshold for `yalign=v`. If the Sample is lower than this (or sample a higher than the 1st line of the Source plus `\vextravskip` – typical for `\includegraphics`), its top will be aligned to the top of the Source, otherwise its top baseline (with `\vtop`) will be aligned to the top baseline of the Source.

`xalign=l` Specifies horizontal alignment (`=l` for left, `=r` for right) of the Sample box (and the separator) within its allocated width for side-by-side display. Please note that `=r` works only with `boxstyle=h`, because all other box sizes use their full allocated width.

`xindent=deeppre` Specifies additional horizontal indentation in display verbatim mode. Use `=none` to get no extra indentation. Use `=narrower` to get `\narrower` (both `\leftskip` and `\rightskip` are decreased by `\parindent`). Use `=deeper` to move one level deeper in the list environment hierarchy and get that indentation. Use `=deeppre` (default) to move one level deeper, but don't change indentation (this is useful with `yindent=deeper` – otherwise it is equivalent to `=none`). Use `=deepright` to set both left and right indentation from the left indentation of `=deeper`.

`yalign=u` Specifies vertical alignment in side-by-side display. By default, the top of the bounding boxes of the Source and the Sample is aligned, which looks nice if the Sample is an image, but doesn't align properly if the Sample is text with a font of similar size to the Source. Use `=b` to align the topmost baselines of the Sample and the Source. This looks nice if the Sample is text, but it is ugly if the Sample is an image higher than `\baselineskip`. The use of `=v` is recommended, which decides between `=b` and `=u` based on the height of the Sample (see `vsmallht=` for the details).

`yindent=deeper` Specifies the vertical space separating display verbatim from the surrounding text. Use `=none` no have no extra vertical space, the display verbatim appears to be a new paragraph as far as `\baselineskip` and `\vskips` are concerned. Use `=deeper` (default) to move one level deeper in the list environment hierarchy (and use the `\parsep` and `\partopsep` etc. specified there). It is recommended to have `yindent=deeper` and `xindent=deeppre` together, so there is no extra horizontal indentation.

`verbatimenv=unchanged` Use `=yes` to change the implementation of the `verbatim` and `verbatim*` environments to use the `PSource` environment.

`vrule=rule` By default, the Source and the Sample are separated with a vertical rule of width `\arrayrulewidth` in the middle of a horizontal space specified by `source-sepwidth=`. Use `=skip` to omit the rule but keep the space. Use `=none` to have no separator at all.

The other packages (codep and verbfwr) shipped with examplep do not have load options.

4.1 Interface to the listings package

The listings package [1] provides advanced typographic for display verbatim, including proper typesetting of tabulators and syntax highlighting for more than a hundred languages. examplep doesn't try to reimplement these features, but it supports calling the listings package to typeset the Source lines in display verbatim. The surroundings (line numbers, vertical separation, horizontal margins and the Sample) are not effected, only the Source line contents are passed to listings. This implies that the border and the background color support provided by the listings package doesn't work with examplep. To use the interface, the listings package must be loaded, and either the listings= or the ttlistings= options of examplep has to be active when the Source is typeset. Additional options can be specified to listings in the argument of `\lstset` at any time. The interface has been tested with the listings package dated 2000/08/23 and 2004/09/07.

examplep treats tabulators (ASCII code 9) as 8 spaces. This is acceptable at the beginning of the line, but it may be incorrect elsewhere. To get tabs right, specify the `ttlistings=yes`, or, to be more precise, the `ttlistings={tabsize=8}` option to examplep. It is also possible to have visible tabulators: specify, for example, `ttlistings={tabsize=8,showtabs}`. In our tests listings failed to detect the width of a character of a fixed width font, so examplep enforces character width using the natural width of the space each time it calls listings. This workaround made the `showtabs` listings option work properly. See the documentation of the listings package for options that affect the typesetting of Source line contents. See an example of using listings from examplep on page 16.

listings supports fixed width characters with a variable with fonts. However, this support seems to be broken when used with examplep, so the `columns=fullflexible` listings option is enforced so proportional fonts will look proportional. Although the listings package claims that it has accented letter support, this didn't work well with the single-character accented letters input using the inputenc package (those characters were positioned to a wrong place inside the line, possibly because listings has failed to recognise that `\lst@UseLostSpace\lst@PrintToken` has to be inserted in front of the accented character into its internal token list). examplep contains a work-around to this problem, with the following limitations: multibyte input encodings such as UTF-8 are not supported (will print strange error message); accented characters may not be part of keyword names in syntax highlighting; accented characters are shown as `^^` hex escapes in aligned mode (see in Subsection 3.2), so they don't work with `\PexaShowBoth`.

listings, when called from examplep, failed to break ligatures such as `‘?` and `<<`. This has the side effect that guillemots would be typeset instead of bitwise right shift in C language sources. examplep modifies the `\lst@FillOutputBox@` macro so it will add a `\relax` between each character displayed – so all ligatures are broken. (This approach is quite differrent from the way LaTeX disables a few ligatures with `\@noligs`; `\pexa@noligs` is similar to `\@noligs` in this respect.)

It is possible to customize the listings package so it typesets some strings differently. For example, with the `literate={<=}{{$\leq$}}1` listings option, all occurences of `<=` (even those inside strings of the target programming language) are typeset as $\leq$. There are no problems when using this feature from within examplep. It is also possible for strings and comments in the syntax-highlighted Source to span multiple lines – listings takes care to remember its internal state between lines.

5 Commands and environments

The arguments between brackets (`[` and `]`) are optional: either the the argument and the brackets are all missing all all present. The arguments named "options" is a comma-separated list of local customization options, defined in Section 4. The `{`$^+$ notation in front of an argument means that the argument can be delimited by braces (thus it must be properly nested), or with any symbol in `\dospecials` (`\ $ & # ^ _ % ~`) or in `\pexa@cverb@donormals` (`` ` ! @ * - + = | : ; ' " , . / ? < > ( ) [ ] ``).

`\PVerb`[⟨options⟩]`{`$^+$⟨verbatimtext⟩`}` Typesets its argument in inline verbatim mode. Similar to the LaTeX `\verb` macro, but respects the options. The use of `[]` is recommended instead of omitting the options altogether, because `[]` will ensure that the proper catcode changes are in effect even for the first verbatim character. This command is robust.

`\PVerbH{`$^+$⟨verbatimtext⟩`}` Shorthand for `\PVerb[pverb-hash=half]` (extra options cannot be specified). This command is robust.

`\PVerbInner\PVerb...` Forces the `\PVerb`... command immediately following it to work in inner mode, thus compressing spaces, respecting comment characters etc. Because of how TEX works, it is impossible to go the other way round, and force outer mode, because it is too late change catcodes – the argument has already been tokenized in inner mode. This command is robust.

`\PVerbOpt{`⟨options⟩`}{`$^+$⟨verbatimtext⟩`}` Equivalent to `\PVerb`, but uses a different syntax. For example, `\item[\PVerb[pverb-space=visbreak]{xy}]` doesn't work because of the nested `[`. Use this instead: `\item[\PVerbOpt{pverb-space=visbreak}{xy}]`, or `\item[{\PVerb[pverb-space=visbreak]{xy}}]`. This command is robust.

`\Q{`⟨verbatimtext⟩`}` Similar to `\PVerb`, but its argument must be escaped (see in Subsection 3.1), and it can be used in section titles etc. Must be enabled with `Q=yes`. This command is robust.

÷⟨verbatimtext⟩÷ Similar to `\PVerb`, but it can be used in section titles etc. (but not int `tabular`) (see in Subsection 3.1). Must be enabled with `div=yes`. This command is robust.

\÷⟨verbatimtext⟩÷ Equivalent to `\Q`, but the argument delimiter is different. Similar to `\PVerb`, but its argument must be escaped (see in Subsection 3.1), and it can be used in section titles etc. Must be enabled with `bsdiv=yes`. This command is robust.

`\url{`⟨url⟩`}` Must be enabled with `url=yes`. This command is robust.

`\begin{WFile}{`⟨filename⟩`}` (defined in the verbfwr package) Writes its contents verbatim to the specified file. TEX `.tcx` and line ending transformations apply, so it is possible that accented letters will be converted to `^^`hex according to the input encoding.

`\begin{WAux}` (defined in the verbfwr package) Writes its contents verbatim into the current `.aux` file. TEX `.tcx` and line ending transformations apply, so it is possible that accented letters will be converted to `^^`hex according to the input encoding.

`\begin{PWSource}`[⟨options⟩] Comination of `\begin{WSource}` and `\PexaShowSource`. It is recommended to have a `[]` even if there are no options, so the very first token of the contents will be read with proper catcodes.

`\begin{WBoth}` Writes its contents to the Source and the Sample temporary file. It is a combination of `WSample` and `WSample`.

`\begin{WSample}` Writes its contents to the Sample temporary file (`pexa-sam.tex`), to be typeset by a subsequent `\PexaShowSample` or `\PexaShowBoth`. The line must end at `\end{WSample}` because of technical reasons.

`\begin{WSource}` Writes its contents to the Source temporary file (`pexa-src.tex`), to be typeset by a subsequent `\PexaShowSource` or `\PexaShowBoth`. It is similar to `\begin{verbwrite}` in the `sverb` package and the `\begin{filecontents}` LaTeX built-in environment. The line must end at `\end{WSource}` because of technical reasons.

`\begin{PIgnore}` Ignores everything up to `\end{PIgnore}`. The environment closer must be at the end of its line. Similar to the `comment` environment in some other packages.

`\begin{PSource}`[⟨options⟩] Typesets its contents in display verbatim. Similar to the LaTeX `\begin{verbatim}` environment, but respects the customization options. It is recommended to have a `[]` even if there are no options, so the very first token of the contents will be read with proper catcodes. This environment is similar to `PWSource`, but it doesn't create a temporary file, so it is faster, `srcstyle=leftboth` (etc.) can be used, and there is no ambiguity between `^^e1` and `á` (etc., see more in Subsection 7.2). Page breaks are allowed between each Source line. (The implementation of this environment is fairly complex compared to `PWSource`.)

`\begin{verbatim}` Equivalent to `\begin{PSource}[]`. Must be enabled with `verbatimenv=yes`.

`\begin{verbatim*}` Equivalent to `\begin{PSource}[source-space=visbreak]`.
Must be enabled with `verbatimenv=yes`.

`\begin{PexaMinipage}`[⟨vbox-type⟩]`{`⟨width⟩`}` Similar to the LaTeX `minipage` environment (and accepts the same arguments), but isolates (concerning section numbers etc.) of its contents from the main document more thoroughly. See Subsection 3.3 for details of isolation.

`\PexaShowBoth{`⟨options⟩`}` Typeets the Source and the Sample side-by-side in display verbatim mode. The Source comes from the temporary file written by the last `WSource` or `WBoth` environment, and the Sample comes from the temporary file written by the last `WSample` or `WBoth` environment. By default, a vertical separator line is drawn between the Source and the Sample, and page breaks are allowed in the Source after the end of Sample. It can be called multiple times with different options for the same file.

`\PexaShowSample{`⟨options⟩`}` Typesets the Sample (written by the last `WSample` or `WBoth` environment) in display mode. It can be called multiple times with different options for the same file.

`\PexaShowSource{`⟨options⟩`}` Equivalent to `\PexaInputSource` with the file written by the last `WSource` or `WBoth` environment. It can be called multiple times with different options for the same file.

`\PexaInputSource{`⟨filename⟩`}{`⟨options⟩`}` Typesets the contents of the specified file as Source in display verbatim mode.

`\begin{code}` (defined in the `codep` package) Typesets its contents side-by-side and also marks its contents to be dumped to the CD. By default, each line is emitted to all three streams, but lines with special prefixes will go into the Source, Sample or CD-file stream only. See Section 6 for details.

`\PexaAllowBreak` Allows a line break here with a discreationary specified in the option `pverb-linebreakchar=` inserted.

`\abreak` A robust command which inserts `\PexaAllowBreak` when the font `{\ttdefault}{m}{n}` is active; inserts `\allowbreak` otherwise. Must be enabled with `abreak=yes`.

6 Writing examples with the codep package

Textbooks and manuals tend to have many display verbatim examples. The examples are usually code snippets which can be further processed by a compiler or another program. Sometimes minor modifications, such as adding the proper header or trailer, are necessary before the code snippet can be processed. It is customary to put all code snippets in the book onto the CD accompanying the book. The `code` environment of the `codep` package (part of the `examplep` distribution) generates CD-files automatically.

Three streams are generated from the contents of each `code` environment: the Source, the Sample and the CD-file streams. Most parts of these streams are identical. The Sample usually differs from the Source because the code snippet has to be typeset specially in the book (for example, `\includegraphics` has to be used to typeset an EPS file whose Source is displayed). The CD-file differs from Source because additional header and footer may be required (such as `\begin{document}` etc.), which are omitted from the book to conserve space.

The `code` environment reads the code snippet line-by-line. The type of the line is specified in first two characters. Lines having the default type are written to all 3 streams, and special line types exist to write to a specific stream only. The `code` environment writes the Source and Sample streams to temporary files, and upon the end of the environment, it calls `\PexaShowBoth` (or `\PexaShowSource`, if the Sample stream is empty) to typeset the example. The CD-file stream is not written to a file by TEX, but the file name and starting line number of the `code` environment is reported in the `.aux` file. A Perl script (wrfiles.pl, part of the examplep distribution) has to be called later to the actual generaton of CD-files. It will examine the `.aux` files, extract the CD-file stream from the `.tex` files, and dump these streams to individual files in the `CDfiles` directory. The file names can be specified in the `code` enviroment, and the environment can generate file names based on chapter and page numbers (so the reader will know from the file name where to read more about the example). The same file name is never generated again.

The code package was used in our recent LATEX textbook [4] to typeset its examples. Most of the examples were written in LATEX, but many of them were METAPOST sources, and some of them were others (e.g. configuration files, shell scripts or EPS files). Because of the huge amount of LATEX examples, special features were added to make them easy and convenient to input for the author. For example,

```
\begin{code}
t \usepackage{url}
  URL:
  \\\url{http://foo.org/~user/}
\end{code}
```

is displayed as (depending on the examplep options)

```
1%^\usepackage{url}            | URL:
2URL:                          | http//foo.org/~user/
3\\\url{http//foo.org/~user/}
```

As seen above, examples are quite convenient to input, and examplep takes care of typesetting side-by-side, determining width of the Source, allowing page breaks, putting margins and `\vskip`s right, adding the rule the separate the Source and the Sample, adding line numbers, generating file name for CD-file and writing the CD-file with header and footer.

With codep it is easy to fulfill the following quality criterias: the Sample must be consistent with the Source (i.e. if the Source is changed during editing to book, the Sample should change automatically); the CD-file must be consistent with the Source; the CD-file must be directly compilable with LATEX (so a header and a footer have to be added). When the deadline of finishing the book approaches, there might not be enough time left to ensure these manually, so a package such as codep is very useful in this situation.

6.1 Example files on the CD

The following CD-file is generated from the code snippet above:

```
\documentclass{article}

\usepackage[latin2]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[magyar]{babel}
\usepackage{url}

\begin{document}

URL:
\\\url{http://foo.org/~user/}

\end{document}
```

The `CodeDefaultD`, `CodeDefaultL`, `CodeDefaultB` and `CodeDefaultE` environments can be used in the preamble to customize the default header and footer generated into the CD-file. For example:

```
\begin{CodeDefaultD}
\documentclass[10pt]{article}
\end{CodeDefaultD}
\begin{CodeDefaultL}
```

```
\usepackage[latin2]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[english]{babel}
\end{CodeDefaultL}
```

Although TEX is able to write to external files with `\textsfrite`, there were several reasons for using an external program (a Perl script) to extract the source snippets from the document sources:

- with `\write` the file always ends at end-of-line
- `\write` forces `.tex` if no extension is specified
- `\write` removes whitespace from end-of-line
- `\write` translates accented letters to hat-escapes (e.g. `á` to `^^e1`) unless compiled with `latex -translate-file cp8bit.tcx` (`-translate-file il2-t1.tcx` makes ő in DVI incorrect). There is the same problem when emitting UTF-8 text.
- it is impossible to distinguish missing files from empty files, so accidental file overwrites are hard to prevent
- it is too late to verbatize if the verbatim text is inside braced macro arguments

The only limitations of this solution are: is not possible to `\input` or `\include` a subfile, and then use the `code` environment in the referrer file; the subfile has to be included with `\include{...}` or `\input{...}` (with braces); and the subfile must have extension `.tex`. The first one is usually not a problem, since referrer files themselves do not typeset text, they only include subfiles. See Subsection 7.7 for implementation details.

6.2 `\begin{code}` invocation

The input syntax of the `code` environment has been designed so that typing the most common examples (short LATEX code snippets) is simple and straightforward, but the author can have full control over all three streams if he wants to. The contents of the environment is divided into lines. The first two characters of each line specify the line type and the rest is the line data. The first character of the line type is usually a lowercase ASCII letter or a punctuation symbol. Line types belong to classes, which are denoted by capital ASCII letters. The order of the classes in the environment is significant, but the order of the individual types or lines within the class is irrelevant. Some classes have default lines, which are used only if the class is omitted from the environment. The default lines make it possible to have default CD-file header and trailer. The clases, in proper order with allowed types in parentheses), are:

F (f, f!, v, v!) specify the file name.

D (d) the `\documentclass` line, default uses article

L (l) the preamble specific to the natural language, defaults for Hungarian babel, Latin-2 inputenc, T1 fontenc. Use the `CodeDefaultL` environment to override.

P (p≡0, t) the preamble with the `\usepackage` lines

B (b) `\begin{document}`

C (<≡c, >≡o, ␣≡2, w, s, x, %) the document contents

E (e) `\end{document}`

The meaning of the complicated types are:

f Accepts a file name with extension. The use of `_` in the name is not recommended. The extension (e.g. `.tex`) is mandatory. The chapter and page numbers will be prepended to the file name (only the page number for document classes without chapters), for example `f foo.mp` may become `2_63_foo.mp` in chapter 2, on page 63.

v Like `f`, but removes the default lines from classes D, L, P, B and E. This is ideal for emitting non-LATEX examples.

f! Like `f`, but don't prepend numbers to the file name.

v! Like `v`, but don't prepend numbers to the file name.

p≡0 Writes only to the preamble of the CD-file.

t Writes to CD-file, appends line prefixed by `%^` to Source. Useful to indicate in the book that a package is needed. Example: `t␣\usepackage{url}`.

<≡c Writes to Source and CD-file.

> Writes only to Sample.

x Writes to Sample and CD-file.

␣≡2 Writes to Source, CD-file and Sample.

w Writes only to CD-file.

s Writes only to Source.

% Comment, ignored.

The `code` environment omits the Sample part from the book if the Sample is empty, and it omits the whole display verbatim environment (but still writes to CD-files) if both the Sample and Source are empty.

6.3 An example with METAPOST code

If the eempost package is also loaded, the following code can be used to typeset a simple, syntax-highlighted METAPOST source and its output:

```
{\PexaDefaults{listings={language=metapost}}\begin{code}
v house.mp
> \begin{EempDef}{house.1}{}{}
w beginfig(1)
  u:=18bp; picture V; V:=image(
    draw unitsquare scaled u xscaled 2;
    fill (0,u)--(2u,u)--(u,1.5u)--cycle
      withcolor red);
  draw V rotated 10;
  draw V shifted (3u,0);
w endfig; end
> \end{EempDef}
> \leavevmode\EempUseFig{house.1}{0}{0}
% ^^^ Dat: \leavevmode to get the Overfull \hbox warning
\end{code}
} % Dat: nothing allowed after \end{code} in its line
```

If eempost is not loaded, the following code should be used instead:

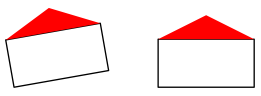
```
{\PexaDefaults{listings={language=metapost}}\begin{code}
v house.mp
> \begin{WFile}{house.mp}
x beginfig(2)
  u:=18bp; picture V; V:=image(
    draw unitsquare scaled u xscaled 2;
    fill (0,u)--(2u,u)--(u,1.5u)--cycle
      withcolor red);
  draw V rotated 10;
  draw V shifted (3u,0);
x endfig; end
> \end{WFile}
> \leavevmode\includemps{house.2}
\end{code}
}
```

The `\includemps` command should be defined in the preamble as:

```
\usepackage{graphicx}
\DeclareGraphicsRule{*}{mps}{*}{}
\makeatletter
\@ifundefined{Ginclude@eps}{}{\def\Ginclude@mps{\Ginclude@eps}}
\def\includemps{\@ifnextchar[\includempsb{\includempsb[]}}
\def\includempsb[#1]#2{\includempsc{#1}#2\@nil}
\def\includempsc#1#2.#3\@nil{%
  \IfFileExists{#2.#3}{\includegraphics[#1]{#2.#3}}{
  \GenericWarning{}{Please run: mpost #2^^J\@gobble}}}
\makeatother
```

This should work with both dvips and pdflatex. The typeset output looks like this:

```
u:=18bp; picture V; V:=image(
 draw unitsquare scaled u xscaled 2;
 fill (0,u)−−(2u,u)−−(u,1.5u)−−cycle
   withcolor red);
draw V rotated 10;
draw V shifted (3u,0);
```



7 Some implementation details

7.1 Starting from poor man's inline verbatim

The following macro, derived from a macro in the `.dtx` documentation of David Kastrup's binhex package [2], typesets its argument in inline verbatim mode:

```
{\catcode\string`>12 \gdef\stripprefix#1>{}}
\def\verbatize#1{{\ttfamily
   \toks0{#1}\edef\next{\the\toks0}% Dat: make # OK
   \fontdimen2\font=0pt % Dat: hide spaces
   \expandafter\stripprefix\meaning\next
   \unskip % Dat: strip final space, possibly after command
   \fontdimen2\font=\dimen0}}% Dat: reset global change
```

This demonstration show how useful the TEX primitives `\string` and `\meaning` are. Both of them convert tokens to characters with catcode 12 (other) or 10 (space). Token lists with spaces are hard to post-process by TEX macros, because TEX macro expansion ignores spaces before undelimited macro arguments. But it is possible to write a macro which converts spaces to anything with catcode 12, for example the `\sca` macro below does this:

```
\begingroup\catcode\string``12 \lccode```\%\lowercase{\endgroup
  \def\scc#1 {\ifx\hfuzz#1\else#1`\expandafter\scc\fi}}
\def\scb#1#2{\scc#2\hfuzz#1} \def\sca{\scb{ }}
% try with: \message{\sca{foo  bar }}
```

It is possible to change `%` in the definition above to anything, including a space: the replacement character will have catcode 12. After such a conversion, the text to be emitted can be easily processed to add TEX macros, change catcodes back to 13 for ISO Latin high accented characters, replace spaces with appropriate constructs, insert `\allowbreak` to the right places to enable line breaks etc. The `\PVerb` macro, when invoked in inner mode (i.e. read inside a macro argument) works this way, and respects the options specified by the author.

7.2 Hex escapes with output translation

The TEX primitives `\write`, `\message` and `\errmessage` may escape some characters when printing them. By default, TEX changes the code ranges 0–31 and 127–255 (the codes outside the printable ASCII range), escaping such codes with a `^^`: for example, the tabulator (code 9) becomes `^^I`, and characters having a high code in the font (not the input) encoding are dumped in hexadecimal, for example ő (having code 174 in T1 encoding) becomes Ž. (This behaviour depends on the default `.tcx` file the TEX distribution uses. No translation occurs with cp8bit.tcx. To spot the difference, run `tex -translate-file cp8bit "\message{^^I^^1fá}\end"`, and then change `cp8bit` to `.missing.`, and run again.) The transformation is lossy: both `\message{ő}` and `\message{\string^^ae}` yield the same result: `^^ae`. Escaping the caret as `^^5e` doesn't help either, because the TEX unescapes carets recursively when reading back the written file. Since ISO Latin accented characters are more often needed in verbatim environments than double carets, examplep does the necessary unescaping when it reads the file back. The back-transformation doesn't work with UTF-8, because the 2nd byte is not decoded by the time the first one is being executed. The unescaping would be done by TEX itself if the caret had its original catcode 7, but that would imply that the non-escaping, verbatim carets wouldn't work.

The unescaping is implemented in a straighforward, but ugly way in the `\pexa@dohex@low`... macros. The caret escapes are parsed in a huge `\if\else\if` construct nested in 40 levels, and once the hexadecimal code is available and converted to upper case, the `\lccode'+="`⟨code⟩`\lowercase{+}` construct is used to insert the appropriate character with catcode 12 (`~` is used instead of `+` to get an active character, catcode 13). The construct is not expandable, but it works because it is used for typesetting. The caret is made active and defined to execute `\pexa@dohex`, so each caret in the file will get unescped.

7.3 Disabling ligatures

The only way to disable a ligature in TEX is to insert a nonexpandable tokens into the input stream between the characters forming the ligature. For example, `f{}i` or `f\relax i` can be used to get "fi" instead of "ﬁ". The most important ligatures (in addition to ligature letters) to be disabled in verbatim mode are: `<< >> ?‘ !‘ ,, ‘‘ ’’ --` and `---`. This can be accomplised by inserting a `\relax` token in front of each `‘ ’ , - <` and `>`. The `\pexa@noligs@some` command of examplep does exactly this, for example, it defines `{\lccode‘~‘<13 \gdef~{\relax\string~}}`. The definition slightly different from the one of the `\@noligs` command in the LATEX kernel: `\def<{\leavevmode\kern\z@\char‘\<}`; but the effect is the same. The `\pexa@noligs@most` command, on the other hand, makes all characters with category code 12 in the range 32. . . 127 active, and adds `\relax` to both sides. This change doesn't affect ASCII or accented letters, but usually there are no ligatures with letters in typewriter fonts. See also the `noligs=` load option.

7.4 Detecting inner/outer brace in inline verbatim mode

The `\PVerb` commands work differently based on whether they are inside a macro argument or not. More precisely, they detect whether they are able to change the catcode of the following token. If so, they are in outer mode (i.e. outside a macro argument), so they change all the other catcodes as well, so consecutive spaces and comment characters will be included in verbatim, too. Otherwise, they are in inner mode, their argument is already read and tokenized by TEX's eyes, so changing catcodes is pointless.

The auto-detection works this way: the catcode of all the special characters (as enumerated in `\dospecials`; including braces) is changed to 3 (math-shift). Then the next token is read into `\reserved@a` with `\afterassignment\pexa@cverb@gottoken\let\reserved@a=` . No tokens are ignored this way, not even spaces. The `\pexa@cverb@gottoken` macro then examines the catcode of the character in `\reserved@a`, and if it is 3, it continues in outer mode, otherwise it continues in inner mode. In inner mode, the next token is forced to be an open-brace, because verbatim material with braces not nested cannot be read into inner mode anyway (TEX would print an error message when it is trying to find the end of the macro argument containing the `\PVerb` construct).

Another common trick is used when parsing the argument in outer mode when it is delimited by braces. Normally a TEX macro expansion (using the definition `\def\pexa@cverb@outerc#1{...}`) can read an argument that is in braces, but in our case the very first opening brace has been already read (by `\let` above), so we have to insert it back: `\catcode‘\{1 \catcode‘\}2 \expandafter\pexa@cverb@outerc\expandafter{\iffalse}\fi`. The `\iffalse}\fi` here is needed for making the definition properly nested.

7.5 Inline verbatim in section titles

The TEX command `\write`, `\message` and `\edef` fully expand their arguments, and similar expansion is enforced by the `\markboth` built-in LATEX macro for section titles and page headings. Therefore macros in section titles have to be protected so their expansion is delayed until the section title is typeset. LATEX offers `\protect` for this: if the macro control sequence is preceded by `\protect`, its expansion is properly delayed; the expansion of the argument has to be delayed manually in a similar way. Some macros have `\protect`ion included; they are called "robust". If the definition a macro starts with `\DeclareRobustCommand` instead of `\newcommand`, the macro is defined to be robust (and its body can be retrieved by looking at the control sequence with a space added, e.g. `\expandafter\show\csname␣sqrt␣\endcsname`).

`\protect` can have three definitions depending on what time it is processed: it is `\string` in a `\typeout` or a LATEX error or warning message (try `\typeout{\meaning\protect}`); it is `\noexpand\protect\noexpand` when `\write`ing to a file (most commonly the `.aux` file); otherwise it is just `\relax` (≡ `\@typeset@protect`; try `\pagestyle{headings}\section{\meaning\protect}` and spot the difference between the main text, the section title and the `.aux` file).

The `\÷` and `\Q` inline verbatim commands are made robust, so they can be used in macro arguments. In fact, they are extra-robust, since they take care of protecting their arguments when being written to a file by LATEX. Protecting here means adding `\noexpand` in front of each token in the argument. The token parsing is easy since the argument – by the nature of these commands – may not contain braces or spaces. The implementation looks like this.

```
\long\def\÷#1÷{\Q{#1}}
\long\def\Q{\ifx\protect\@typeset@protect\expandafter\@gobble\fi
  \@thirdofthree\@firstoftwo\displayit\protectit}
\def\displayit#1{...}
\def\protectit#1{\noexpand\÷\protectnext#1÷}
\long\def\protectnext#1{\noexpand#1%
  \ifx#1÷\else\expandafter\protectnext\fi}
```

The first trick is in the body of `\Q`: the argument is passed to either `\displayit` or `\protectit`, depending on the current value of `\protect`. If the condition is true, `\@gobble` is called, which removes `\@thirdofthree`, so `\@firstoftwo` will choose `\displayit` (otherwise, `\@thirdofthree` chooses `\protectit`). The second, more classical trick is the rôle of `\expandafter` in the definition of `\protectnext`: it makes the `\fi` token disappear, so the tail-recursive call to `\protectnext` will grab the next token into `#1` instead of `\fi` itself.

7.6 Special hyphenchar in inline verbatim

When inline verbatim is hyphenated, care has to be taken to make the discretionary hyphen different from a regular, verbatim hyphen. (There is a similar problem with spaces disappearing when the line is broken; to avoid this, try setting the option `pverb-space=visbreak` or `pverb-space=invdisc`.) TEX auto-hyphenation takes the discretionary hyphen from the `\hyphenchar` of the font. So the solution is adding a new glyph to the verbatim font, changing the font encoding vector to include the glyph, and then setting `\hyphenchar`.

We have chosen character position 24 (per-thousand sign) of the T1 encoding to be replaced by a soft hyphen (¬), which is deliberately narrower than all the other characters, so the reader immediately sees its function. For example: "`foo¬`
`bar`". We have drawn the glyph in Fontforge, saved the data to PFB, converted it to human-readable format with the command `type1fix.pl shorthyp.pfb gsx:  shorthyp.gsx`, extracted the human readable glyph definition (`/shorthyp { ... }`) from the output. We have changed the `/FontName` and injected the glyph to original font with the following command:

```
perl -x -S type1fix.pl --set-leniv=0 --dump-spaces=no --pack \
  --dump-bars --dump-stde --dump-ends=no --debug-warnings \
  --chk-insize=no --set-uniqueid=random --set-fontname=t1xtts \
  --set-glyph="/shorthyp { 50 354 hsbw 315 vmoveto -17 vlineto 0
  -8 0 -8 6 -4 rrcurveto 4 -6 8 0 5 0 rrcurveto 195 hlineto -124
  vlineto -11 0 -21 15 vhcurveto 2 0 3 1 2 1 rrcurveto 10 2 1 12
  0 12 rrcurveto 0 8 -1 8 0 5 rrcurveto 130 vlineto 0 5 1 7 0 7
  rrcurveto 0 12 -2 11 -11 3 rrcurveto -5 1 -6 0 -5 0 rrcurveto
  -12 0 -12 -1 -10 0 rrcurveto -98 hlineto -16 0 -19 2 -17 0
  rrcurveto -32 -6 -3 -24 hvcurveto closepath endchar} def" \
  t1xtt.pfb pfb: t1xtt-shorthyp.pfb
```

We have changed six lines in tex256.enc to match the glyph names in the font (e.g. `/endash` → `/rangedash`), and we have changed position 24 to `/shorthyp`. We have also changed the name of the encoding in the beginning of the file. We have inserted the following line to the PostScript font map files (e.g. psfonts.map), without the line break:

```
t1xtts t1xtts "TeX256-shorthypEncoding ReEncodeFont"
  <tex256-shorthyp.enc <t1xtt-shorthyp.pfb
```

We have also added a new TFM file based on the old one. We have dumped the old one with `tftopl -charcode-format=octal t1xtt.tfm`, modified the width (`CHARWD`) of character 24 (`CHARACTER O 30`), and saved the modifications with `pltotf modified.pl t1xtts.tfm`. We've added the LATEX font map file t1xtts.fd with the following content:

```
\DeclareFontFamily{T1}{xtts}{\hyphenchar\font\m@ne}
\DeclareFontShape {T1}{xtts}{m}{n}{<->t1xtts}
```

The `\hyphenchar` settings above disables automatic word hyphenation, so words inside `\texttt` etc. won't be accidentally hyphenated. We have copied all the files above to the appropriate directories and we have run mktexlsr to update the file list. We have included some options in `\PexaDefaults` line in the document preamble: `pverb-hyphenchar=char'30` (for automatic word hyphenation) `pverb-linebreak=char`, `pverb-linebreakchar={\string\char'30␣}` (inserted around symbols).

We have also defined `\def\pexa@verbatimfont{\normalfont\fontfamily{xtts}\selectfont}`, and we have made sure that the T1 encoding is in use (`\usepackage{t1enc}`).

The overall effect of these modifications was that examplep now used our glyph for automatic word hyphenation and as discreationary hyphen around symbols in inline verbatim mode. The demonstrations above shows that it is quite complicated to change a single glyph in a LATEX font. It is hoped that the situtation will improve with TEX's successors.

7.7 Passing information about the CD-files to wrfiles.pl

wrfiles.pl is used to extract the CD-files from the LATEX sources of a book. The reasons why an external program is used instead of TEX's built-in `\write` command are described in Subsection 6.1.

The file names and environment start line numbers are passed to wrfiles.pl in the `.aux` file(s). For example, the line `\@gobble{code:foo.tex:156:2_pic3.mp}` is a declaration that there is a `code` environment starting at line 156 in the file `foo.tex`. wrfiles.pl understands such declarations, and it also understands lines like `\@input{foo1.aux}`, so dumping works even if the document is separated to several `\include`d source files. The declaration above is ignored by LATEX when it reads back the `.aux` file (because `\@gobble` gobbles its argument).

Although the `\inputlineno` primitive is mentioned twice in the TEXbook [3], its – rather straightforward – purpose is not documented there. But the real problem is that TEX doesn't remember the name of the file being read. `\jobname` contains the name of the top-level `.tex` file, so it doesn't work when that file `\include`s or `\input`s subfiles containing `code`. The codep package thus modifies the `\InputIfFileExists` command to save the file name to the macro `\codep@code@@inputfile` if the extension is `.tex`. (The other most common extension after the preamble is `.fd`: such a file is loaded each time a LATEX font that has not been used yet is selected.) The implicit limitation here that `code` won't work unless the extension of the file included is `.tex`. Hooking `\InputIfFileExists` affects `\include{...}` and `\input{...}`, but not not `\input␣...`, `\documentclass` or `\usepackage`. This is not a problem if the author remember that he has to use braces around the file name.

Since there is no hook for `\endinput` (and some packages rely on that `\endinput` is an expandable primitive), it is not possible to set up a stack of names of files being read. Thus, if file *A* has included file *B*, an after that `code` environment placed in *A* will not work, because the declaration line read by wrfiles.pl will contain the name of *B* instead of *A*. This is not a serious limitation, becase files including other files usually don't typeset text by themselves after the inclusion.

The primary reason why wrfiles.pl needs the `.aux` file is that it has to embed the page and chapter numbers into the file names. Although wrfiles.pl could find the source file with the `code` environments by trying to match line numbers with all source files in the current directory, we have decided to make it fail when the file name is not emitted properly into the declaration, so it is sure that the examples in the book and on the CD are consistent.

8 Future work

The most important features to be added and other improvement possibilites:

- a better approach towards automatic hyphenation of inline verbatim, after studies in typography
- allow wider sample if source is small enough
- why doesn't `\selectlanguage` work inside `\begin{PSource}[srcstyle=leftboth]`
- `\PVerb{foo}` mustn't insert "¬" if foo is at end-of-line
- `\PVerb{...}` inner unnested braces (`\futurelet`?)
- differentiated `\penalty` values in `\PVerb`
- paragraph mode should work with side-by-side displays (of course, measuring the width of the Source has still to be done in aligned mode)
- ASCII tabulator (9) characters aren't supported properly, they are just converted to spaces. The width of the tab character should depend on its horizontal position in the line. (With `listings=`, the results are already correct.)
- framing and background color support to display verbatim
- accented characters should work with listings and `\PexaShowBoth`. The original catcode of `^` should be kept so TEX itself would parse the hex escapes.
- an interface to `\lstinline` in listings, with line breaks allowed

9 Conclusion

examplep, as it is now, is a highly customizable LATEX package that provides both inline and display verbatim mode with several advanced features, many of which are not available in any other packages. The code environment is also provided which can typeset both the Source and the Sample column of a side-by-side display verbatim from the same LATEX source stream, furthermore it can emit the stand-alone working version of the Source into a CD-file. These features make the code environment especially useful for sofware textbook and manual authoring. The whole examplep distribution is under the GNU GPL, and it is freely available from CTAN. An earlier version of the packages was used to typeset all the examples in a 770-page introductionary book about LATEX.

examplep is not complete. Some important features are not implemented yet and the package has not been tested thoroughly. Some parts of the code are really ugly, partially because it has not been polished up after writing, and partially because the architecture of TEX and LATEX doesn't provide an elegant way to address the problem. For example, active characters are overloaded: they are used by inputenc, babel (shorthands) and listings (syntax highlighting) for different purposes – these packages have to make extra effort to cooperate with each other. We hope that TEX's successors will improve these conditions, and the core system will provide a generic way to tokenize verbatim text instead of changing catcodes.

References

[1] Carsten Heinz. *The Listings Package*, 7 September 2004.
CTAN:`macros/latex/contrib/listings/listings-1.3.dtx`.

[2] David Kastrup. *The `binhex.tex` package for expansible conversion into binary-based number systems*, 2001.
CTAN:`macros/generic/kastrup/binhex.dtx`.

[3] Donald E. Knuth. *The TEXbook*. Addison–Wesley, 1984.

[4] Ferenc Wettl, Gyula Mayer, and Péter Szabó. *LATEX kézikönyv*. Panem, Budapest, 2004.

[5] Timothy Van Zandt, Denis Girou, and Sebastian Rahtz. *The 'fancyvrb' package. Fancy Verbatims in LATEX*, 1998.
CTAN:`macros/latex/contrib/fancyvrb/fancyvrb.dtx`.

From RTF to XML to LATEX

Andre Dierker Arne Jans
Stephan Lehmke
QuinScape GmbH, Thomasstraße 1, 44135 Dortmund, Germany
{Andre.Dierker,Arne.Jans,Stephan.Lehmke}@QuinScape.de
`http://www.QuinScape.de`

February 26, 2005

Abstract

This paper shows how the widely used Rich Text Format (originally specified by Microsoft) can be processed to produce XML and how the resulting XML-file can be processed by LATEX to produce print-quality PDF-files. To convert the RTF-files to a specific XML-format we use the open-source-tool Majix which can be found on Sourceforge.

We then take XMLTEX to process the generated XML-file. We had to create an output that is as near as possible to Word's. An effort was made to reach this goal even with constructs such as lists, tabstops and especially tables.

Using XML as an intermediary format in typesetting RTF has the advantage that structural transformations are much easier based on XML even if the XML 'only' reproduces the RTF as faithfully as possible.

Filtering or transforming certain objects or attributes and even correcting typesetting errors can be done by appropriate transformations of the XML files.

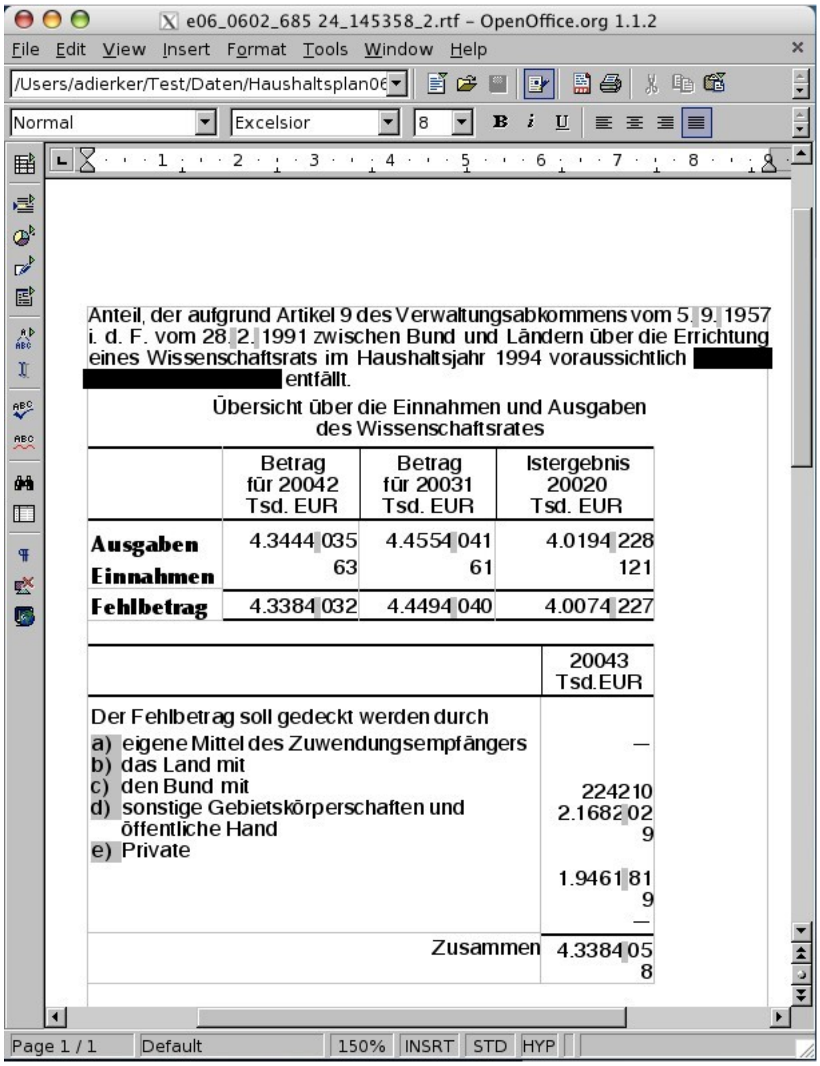
1 Introduction

For a commercial project it was necessary to typeset larger documents from automatically generated XML data with embedded references to external RTF files. It was decided to first transform the RTF to XML and include that into the existing XML-structure, all together then being typeset by XMLTEX.

For this, the open-source-tool Majix was extended to achieve the RTF-XML translation. For typesetting the resulting XML an appropriate implementation using XMLTEX was created which will be uploaded to CTAN eventually, providing another open-source way of handling RTF with TEX.

2 Results

We begin with a RTF-File. As you can see in figure 2 OpenOffice isn't able to handle recorded changes to the file correctly. For example in 'Betrag für 20042 Tsd EUR' (first row, second column) the '20042' is meant to be a year. Originally is was 2002 but the last '2' was deleted


e06_0602_685 24_145358_2.rtf – OpenOffice.org 1.1.2
File Edit View Insert Format Tools Window Help
/Users/adierker/Test/Daten/Haushaltsplan06
Normal
Excelsior
8
Page 1 / 1
Default
150%
INSRT
STD
HYP

Anteil, der aufgrund Artikel 9 des Verwaltungsabkommens vom 5. 9. 1957 i. d. F. vom 28. 2. 1991 zwischen Bund und Ländern über die Errichtung eines Wissenschaftsrats im Haushaltsjahr 1994 voraussichtlich [redacted] [redacted] entfällt.

Übersicht über die Einnahmen und Ausgaben des Wissenschaftsrates

	Betrag für 20042 Tsd. EUR	Betrag für 20031 Tsd. EUR	Istergebnis 20020 Tsd. EUR
Ausgaben	4.3444 035	4.4554 041	4.0194 228
Einnahmen	63	61	121
Fehlbetrag	4.3384 032	4.4494 040	4.0074 227

	20043 Tsd.EUR
Der Fehlbetrag soll gedeckt werden durch	
a) eigene Mittel des Zuwendungsempfängers	–
b) das Land mit	
c) den Bund mit	224210
d) sonstige Gebietskörperschaften und öffentliche Hand	2.1682 02 9
e) Private	1.9461 81 9
	–
Zusammen	4.3384 05 8

Figure 1: We first have a RTF file. Note the mixed up old and new version of the years and amounts. (Please ignore the font type. The used one wasn't installed and had to be substituted by OpenOffice)

and replaced with '4'. However the '2' remained as old version in the RTF and and is tagged as 'deleted'. In opposite to Majix OpenOffice doesn't recognise the responsible control word.

The corresponding RTF-Code is shown in figure 2. We have selected a quite readable portion. When it comes to font management RTF isn't readable at all.

After the conversion by Majix we get the XML-Code shown in table 2. Majix did a great job in giving the data a meaningful structure.

The resulting PDF is shown in figure 2. As you can see there are some differences between the OpenOffice and the LaTeX version. These are the result of some filtering on the XML data demanded by our client.

3 Filtering

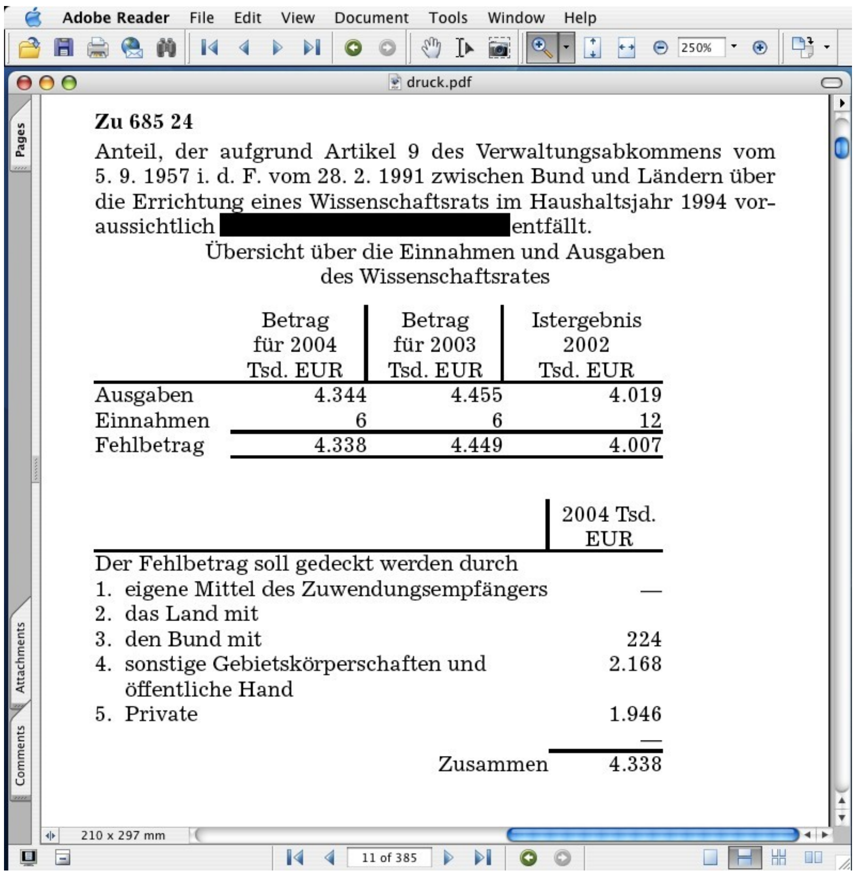
Having a well-formed XML one can easily filter the data. For example harmonizing the indentation of unordered lists can be done by deleting the necessary attributes in the XML and

```
%
\s16\qj\sa40\widctlpar\adjustright \f18\fs16\lang1031\cgrid {\expnd0\expndtw-2 Anteil, der
aufgrund Artikel 9 des Verwaltungsabkommens vom 5.\~9.\~1957 i. d. F. vom 28.\~2.\~1991 zwi
schen Bund und L\'e4ndern \'fc
ber die Err
ichtung eines Wissenschaftsrats im Haushaltsjahr 1994 voraussichtlich entf\'e4llt.
\par }\pard
\plain \s18\qc\sa40\widctlpar\adjustright \f18\fs16\lang1031\cgrid {\'dcbersicht \'fcber di
e Einnahmen und Ausgaben\line des Wissenschaftsrates
\par }\trow
d \clvertalt\clbrdrt\brdrs\brdrw20 \clbrdrb\brdrs\brdrw20 \cltxlrtb \cellx1021\clvertalt\cl
brdrt\brdrs\brdrw20 \clbrdrb\brdrs\brdrw20 \clbrdrr\brdrs\brdrw20 \cltxlrtb \cellx2042\clve
rtalt\clbrdrt\brdrs\brdrw20 \clbrdrb\brdrs\brdrw20 \cltxlrtb
\cellx3063\clvertalt\clbrdrt\brdrs\brdrw20 \clbrdrl\brdrs\brdrw20 \clbrdrb\brdrs\brdrw20 \c
ltxlrtb \cellx4253\pard\plain \qc\sb40\sa40\widctlpar\intbl\adjustright \f18\fs16\lang1031\
cgrid {\cell Betrag\line f\'fcr 200}{\revised\revauth1\revdttm1182249745
4}{\deleted
\revauthdel1\revdttmdel1182249745 2}{\line Tsd. EUR\cell }\pard \qc\fi1\li-1\sb40\widctlpar
\intbl\adjustright {Betrag\line f\'fcr 200}{\revised\revauth1\revdttm1182249745 3}{\deleted
\revauthdel1\revdttmdel1182249745 1}{\line Tsd. EUR\cell
Istergebnis
\line 200}{\revised\revauth1\revdttm1182249745 2}{\deleted\revauthdel1\revdttmdel1182249745
 0}{\line Tsd. EUR\cell }\pard \widctlpar\intbl\adjustright {\row }\trowd \clvertalt\cltxlr
tb \cellx1021\clvertalt\cltxlrtb \cellx2042
\clvertalt\
cltxlrtb \cellx3063\clvertalt\cltxlrtb \cellx4253\pard\plain \s1\sb80\sa40\keepn\widctlpar\
intbl\outlinelevel0\adjustright \b\fs20\lang1031\cgrid {\f18\fs16 Ausgaben}
```

Table 1: The RTF-Code looks like this

```
<par align="justified">
  <tabdeflist>
    <tabdef type="default" align="left" position="12.49mm"/>
  </tabdeflist>
  <parcontent>
    Anteil, der aufgrund Artikel 9 des Verwaltungsabkommens vom 5. 9. 1957 i. d. F.
    vom 28. 2. 1991 zwischen Bund und Ländern über die Errichtung eines
     Wissenschaftsrats im Haushaltsjahr 1994 voraussichtlich entfällt.
  </parcontent>
</par>
<par align="center">
  <tabdeflist>
    <tabdef type="default" align="left" position="12.49mm"/>
  </tabdeflist>
  <parcontent>
    Übersicht über die Einnahmen und Ausgaben<linebreak/>des Wissenschaftsrates
  </parcontent>
</par>
<table>
  <tbody>
    <tr>
      <td width="18.0093mm" valign="top" border-top="0.0pt"
          border-bottom="1.0pt" border-left="0.0pt">
      </td>
      <td width="18.0093mm" valign="top" border-top="0.0pt"
          border-bottom="1.0pt" border-right="1.0pt">
        <par align="center">
          <tabdeflist>
            <tabdef type="default" align="left" position="12.49mm"/>
          </tabdeflist>
          <parcontent>
            Betrag<linebreak/>für 2004<linebreak/>Tsd. EUR
          </parcontent>
        </par>
      </td>
      <td width="18.0093mm" valign="top" border-top="0.0pt"
          border-bottom="1.0pt">
        <par align="center">
          <tabdeflist>
            <tabdef type="default" align="left" position="12.49mm"/>
          </tabdeflist>
          <parcontent>
            Betrag<linebreak/>für 2003<linebreak/>Tsd. EUR
          </parcontent>
        </par>
      </td>
      ...
    </tr>
```

Table 2: The generated XML is much more readable



Zu 685 24

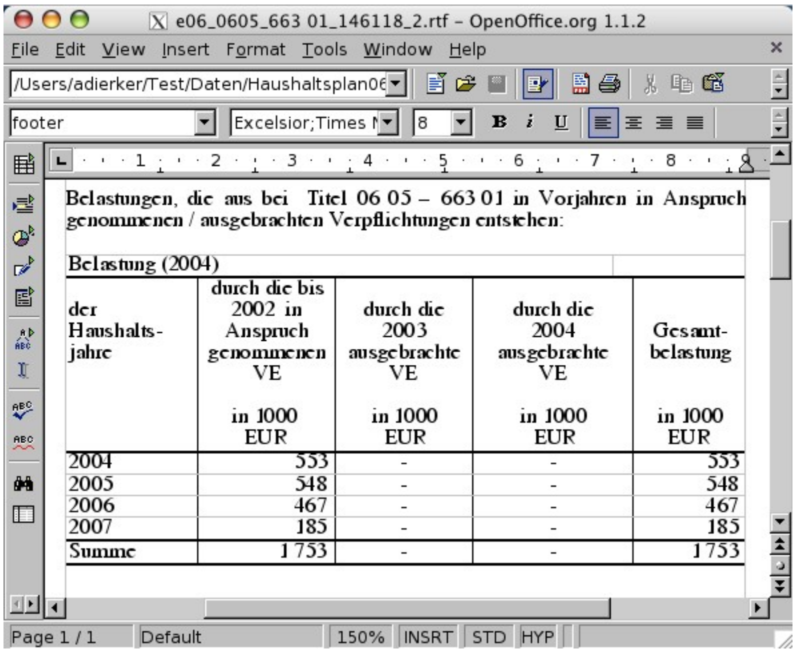
Anteil, der aufgrund Artikel 9 des Verwaltungsabkommens vom 5. 9. 1957 i. d. F. vom 28. 2. 1991 zwischen Bund und Ländern über die Errichtung eines Wissenschaftsrats im Haushaltsjahr 1994 voraussichtlich [redacted] entfällt.

Übersicht über die Einnahmen und Ausgaben des Wissenschaftsrates

	Betrag für 2004 Tsd. EUR	Betrag für 2003 Tsd. EUR	Istergebnis 2002 Tsd. EUR
Ausgaben	4.344	4.455	4.019
Einnahmen	6	6	12
Fehlbetrag	4.338	4.449	4.007

	2004 Tsd. EUR
Der Fehlbetrag soll gedeckt werden durch	
1. eigene Mittel des Zuwendungsempfängers	—
2. das Land mit	
3. den Bund mit	224
4. sonstige Gebietskörperschaften und öffentliche Hand	2.168
5. Private	1.946
	—
Zusammen	4.338

Figure 2: After processing with XMLTEX


e06_0605_663 01_146118_2.rtf – OpenOffice.org 1.1.2
File Edit View Insert Format Tools Window Help
/Users/adierker/Test/Daten/Haushaltsplan06
footer
Excelsior;Times N
8
Page 1 / 1
Default
150%
INSRT
STD
HYP

Belastungen, die aus bei Titel 06 05 – 663 01 in Vorjahren in Anspruch genommenen / ausgebrachten Verpflichtungen entstehen:

Belastung (2004)				
der Haushalts- jahre	durch die bis 2002 in Anspruch genommenen VE in 1000 EUR	durch die 2003 ausgebrachte VE in 1000 EUR	durch die 2004 ausgebrachte VE in 1000 EUR	Gesamt- belastung in 1000 EUR
2004	553	-	-	553
2005	548	-	-	548
2006	467	-	-	467
2007	185	-	-	185
Summe	1 753	-	-	1 753

Figure 3: A special table called VEGrid in a RTF file

providing corresponding defaults.

Another class of transformations deals with the enrichment of semantics by converting visual markup to logical markup: one could search for special XML-constructs perhaps with specific attributes and/or contents and replace them with a 'meaningful' structure.

A good example for this are the so called VEGrids. These are special tables which always have the same layout. A typical VEGrid is shown in figure 3. The headline is always the same as is the tablehead. In the first column there are years, the bottom row is a sum.

Usually VEGrids are already given in a XML-structure but sometimes it seems the person dealing with the case doesn't use the right program to input the data but uses Word to create a RTF file with a VE-lookalike. After the XML conversion we get a noname-table as shown in table 3. Because of the constant layout we could identify these lookalikes and exchange them by 'real' VEGrids as shown in table 3.

4 Further Development

Because there was no need up to now we ignore changes of font size and type. Besides that the management of colours wasn't implemented yet. Perhaps these features will be implemented in future.

As said before it is planned to release the code of the package on CTAN. The extended version of the RTF converter Majix is already available via SourceForge.

```
<table>
  <tbody>
    <tr border-bottom="0.5pt" border-left="0.5pt" border-right="0.5pt"
        border-horizontal="0.5pt" border-vertical="0.5pt">
      <td width="72.5135mm" valign="top" border-top="0.0pt"
          border-left="0.0pt" colspan="4">
        <par align="left">
          <tabdeflist>
            <tabdef type="default" align="left" position="12.49mm"/>
          </tabdeflist>
          <parcontent>Belastung (2004) </parcontent></par></td>
      <td width="17.4978mm" valign="top" border-top="0.0pt"
          border-right="0.0pt" colspan="2"></td>
    </tr>
    <tr border-top="0.5pt" border-bottom="0.5pt" border-left="0.5pt" ...>
      <td width="17.5154mm" valign="top" border-top="1.0pt" ...>
          ...
          <parcontent>
            <linebreak/>der<linebreak/>Haushalts-<linebreak/>jahre
          </parcontent></par></td>
      ...
    </tr>
    <tr border-top="0.5pt" border-bottom="0.5pt" border-left="0.5pt"...>
      <td width="17.5154mm" valign="top" border-top="1.0pt"...>
          ...
          <parcontent>2004</parcontent></par></td>
      <td width="18.1151mm" valign="top" border-top="1.0pt"...>
          ...
          <parcontent>553 </parcontent></par></td>
      <td width="18.1328mm" valign="top" border-top="1.0pt"...>
          ...
          <parcontent>-</parcontent></par></td>
    ...
    </tr>
    <tr border-top="0.5pt" border-bottom="0.5pt" border-left="0.5pt"...>
      <td width="17.5154mm" valign="top" border-top="1.0pt"...>
          ...
          <parcontent>Summe</parcontent></par></td>
      <td width="18.1151mm" valign="top" border-top="1.0pt"...>
          ...
          <parcontent>1 753 </parcontent></par></td>
      ...
    </tr>
  </tbody>
</table>
```

Table 3: The generated XML (note the numerous omissions)

```
<VEGRID>
  <VEROW>
    <VECOLUMN1>2005</VECOLUMN1>
    <VECOLUMN2>553</VECOLUMN2>
    <VECOLUMN3>null</VECOLUMN3>
    <VECOLUMN4>null</VECOLUMN4>
    <VECOLUMN5>553</VECOLUMN5>
  </VEROW>
  <VEROW>
    <VECOLUMN1>2004</VECOLUMN1>
    <VECOLUMN2>548</VECOLUMN2>
    <VECOLUMN3>null</VECOLUMN3>
    <VECOLUMN4>null</VECOLUMN4>
    <VECOLUMN5>548</VECOLUMN5>
  </VEROW>

  ...

  <VESUMROW>
    <VECOLUMN1>Summe</VECOLUMN1>
    <VECOLUMN2>1753</VECOLUMN2>
    <VECOLUMN3>null</VECOLUMN3>
    <VECOLUMN4>null</VECOLUMN4>
    <VECOLUMN5>1753</VECOLUMN5>
  </VESUMROW>
</VEGRID>
```

Table 4: The enriched and filtered XML

TEX forever!

Jonathan Fine
Learning and Teaching Solutions
The Open University
Milton Keynes
United Kingdom
J.Fine@open.ac.uk
http://www.pytex.org

Abstract

This paper explores new ways of doing input to and output from TEX. These new ways bypass our current habits, and provide fresh opportunities.

Usually, TEX is run as a batch program. But when run as a daemon, TEX can be part of an interactive program. Daemons often that run forever, or at least for a long time. Hence the title of this paper.

Usually, parsing and transformation of the input data is done by TEX macros, with little outside help. Often, this results in input documents that only TEX can understand. Also, TEX macros can be hard to write. We demonstrate the replacement of TEX macros by an external program. This is done in real time.

Usually, TEX's principal output is a `dvi` representation of typeset pages, for processing by a printer driver. However, TEX's log file or console can be used to allow TEX to output the boxes it holds internally. (Alternatively, an extension of TEX could write this data out in a binary form.) Shipping out boxes rather than `dvi` allows an external program to do the page makeup.

Don Knuth's original conception was that TEX would be "just a typesetting language". In some sense he "put in many of TEX's programming features only after kicking and screaming". The developments described above reduce our dependence on TEX macros, and so bring our use of TEX closer to Knuth's original conception. Doing this will greatly improve its usefulness.

Long live TEX!

Introduction

In 1990, Don Knuth told us [8, p.572] that his work on developing TEX had come to an end. He went on to say:

> Of course I do not claim to have found the best solution to every problem. I simply claim that it is a great advantage to have a fixed point as a building block. Improved macro packages can be added on the input side; improved device drivers can be added on the output side.

In this paper, the author tries to follow this advice. There are imperfections in TEX, and the lack of proper support for Unicode fonts and filenames is a major weakness. However, TEX also has enormous strengths. It is archival. It carefully uses integer arithmetic to ensure that it gets the same line and page breaks, regardless of the machine it is running on. Its algorithm for breaking a paragraph into lines is reliable, adaptable and efficient. TEX is without rival for complex mathematical typesetting.

Often, TEX is used with LATEX as the macro package front end, and with `dvips` as the device driver. Sometimes, the word 'TEX' is used to refer to the whole system. However, in this paper we mean by 'TEX' the typesetting program written by Don Knuth. And so LATEX and `dvips` are tools for use with TEX.

This paper is concerned with making improvments on the input and output sides of TEX, both areas of work where there is an enormous amount still to do. However, our proposals are not exactly macro packages and device drivers.

A note to the reader: This paper has been written for a general audience, and in particular for those who are not TEX experts. At the same time, discussion of technical details is at times either unavoid-

1	¶	textfile → marked up text and math
2	†	text + transform → horizontal primitives
3	*	horizontal primitives → hlist
4	†	math + transform → math primitives
5	*	math primitives + parameters → hlist
6	*	hlist + parameters → vlist
7	†	vlists + page make up → page boxes
8	*	page boxes → sequential `dvi` file
9	¶	sequential `dvi` file → random access `dvi` file
10	¶	random access `dvi` file → rendered page

Table 1: How TEX works, in 10 stages
† usually done using TEX macros.
* usually done using TEX's built in procedures.
¶ file input and output matters.

able or helpful. Therefore, I hope that the experts will forgive my stating the obvious, and that the others forgive my discussing the difficult.

References: Many of the articles cited here have been reprinted in the collection *Digital Typography* [13]. Page numbers in citations refer to [13], and not to the original publication.

How TEX works

Table 1 gives a concise description as to how TEX works. On the input side we propose that an external program perform the transformation in steps 2 and 4.

On the output side we have two proposals. The first is that (8) be replaced by:

8′. page boxes → stream of `dvi` pages

The second, which is more ambitious, is that (7) be replaced by:

7′. vlist → external program

followed by page makeup in that external program.

Thus we continue to use TEX's excellent typesetting, but reduce the use of its macros.

TEX — just a typesetting language

TEX is a typesetting program, written by Don Knuth, that is particular good at mathematical and technical typesetting. TEX is reliable and stable, and is very widely used by academic mathematicians and physicists.

TEX has a macro programming language, which allows features to be added. The best known and most widely used TEX macro package is LATEX. (This is not quite accurate. Although originally LATEX used TEX, since 2003 it by default uses e-TEX, which is an extension of TEX. So it is no longer purely a TEX macro package. This has no bearing on our discussion.)

In 1996 Don Knuth, describing his intentions when he started to develop TEX, said [11, p.648]:

> I'm not going to design a programming language; I want to have just a typesetting language.

and at the same time he said (loc. cit.):

> In some sense I put in many of TEX's programming features only after kicking and screaming. [. . .] In the 70s, I had a negative reaction to software that tries to be all things to all people. Every system had its own universal Turing machine built into it somehow, and everybody's machine was a little different from everybody else's.

But the need for more features caused the programming constructs to grow (see Table 2 below). See also [16] for a 'wish-list' of future developments.

Therefore, by *removing* commands from TEX, we can come closer to Don's original conception of TEX. However, for this to succeed in practice, some other means of adding new features is required. Indeed, one of the major problems TEX users have now is that the existing programming constructs barely support the demand for new features. This we discuss later.

In this section we outline how to cut TEX down to the bare minimum. To be specific, in this section we ask: What commands are required in order to access TEX's algorithm for breaking a paragraph into lines?

To create a paragraph one needs to be able to load fonts, change fonts, and set a character in the current font. One also needs commands for appending glue, kerns and the like to the paragraph.

To break the paragraph into lines, one needs the `\par` primitive (also known as `\endgraf`) and a means of assigning values to the line-breaking parameters, such as `\hsize`.

In other words, the basic operations are to add an item to the current horizontal list, and to form a paragraph out of the current horizontal list. (For mathematics and table typesetting there are similar basic operations.)

It should at this point be clear that certain primitive TEX commands are *not* required in order to do typesetting. These commands include all the `\def` commands (such as `\def`, `\chardef`, `\xdef`), `\let`, `\begingroup` and `\endgroup`. Once category codes have been set up, there is no further need for `\catcode`. And there is certainly no need for commands such as `\expandafter`, `\noexpand`, `\aftergroup` and `\futurelet`. All these are *not*

Control sequence	Date added
`\if`	21 Jun 1978
`\pausing`	16 Mar 1978
`\uppercase`	25 Nov 1978
`\xdef`	28 Nov 1978
`\ifmmode`	23 July 1978
active characters	25 Jan 1980
`\let`	25 Mar 1980
`\ifx`	13 July 1981
`\catcode`	July 1982
`\expandafter`, `\openin`	12 Sep 1982
`\string`	12 Sep 1982
`\immediate`	12 Oct 1982
`\csname`, `\endcsname`, `\fi`	13 Nov 1982
`\everymath`, `\everydisplay`, `\futurelet`	2 Dec 1982
`\endinput`	7 Dec 1982
`\jobname`	25 Dec 1982
`\globaldefs`	20 Jan 1983
`\iffalse`, `\iftrue`	3 Feb 1983
`\everyvbox`, `\everyhbox`	6 Mar 1983
`\everyjob`	18 Mar 1983
`\advance`, `\multiply`, `\divide` `\noexpand`, `\meaning`	25 May 1983
`\afterassignment`	27 May 1983
`\escapechar`, `\endlinechar`	4 Jul 1983
`\errhelp`	11 Jul 1983
`\aftergroup`, `\newlinechar`	16 Jul 1983
`\ifhbox`, `\ifvbox`	27 Aug 1983
`\holdinginserts`	30 Sep 1989

Table 2: Some TEX control sequences not needed for typesetting (after [7])

typesetting commands, and exist only to allow features to be added to TEX.

Moreover, these commands cause difficulty for both TEX users and programmers. Their introduction is perhaps a sign that things were starting to go in the wrong direction.

In [7] Knuth published, in edited form, the log books he kept while he was developing TEX. In these, we can see the introduction of features. (See Table 2.)

Suppose all programming commands are disabled by `\let`-ting them be undefined, like so:

```
\let \afterassignment = \undefined
```

Provided we remove enough commands, we will have, as Don wanted TEX to be in the first place, “just a typesetting language”. A language without features, and without the capability of adding features (which is itself a feature).

Comparison with PostScript and with machine code is instructive. Most PostScript is generated by programs that translate from a higher-level language down to PostScript. Similarly, much machine code is generated by compiling ‘C’ source files.

Many of us write input files for (LA)TEX, using a text editor. We won’t do that for a featureless TEX. It’s too much hard work, and anyway we want to write in a higher-level language. We are suggesting that an external program perform the text transformation that is traditionally performed by a TEX macro package.

Improved macros — input transformation

This section could also be titled:

```
\let \def = \undefined
```

Don suggested that we add improved macro packages on the input side. Now, a macro package has two main purposes. One is issuing typesetting instructions to TEX. This will create a galley (or page of unlimited depth). The second purpose is the output (or page makeup) routine, which breaks the galley into pages of a suitable size.

In this section we consider the creation of a galley. Marked-up text, such as

```
\section{Improved macros}
```

is translated (by LATEX in this case) into a large number of low-level instructions. The title

```
Improved macros
```

is scarcely translated. Each character sets itself, and space characters produce default interword glue. (Later, we present an example of this.)

It is `\section{}` that does most of the work. Here are some of the technical details. It selects the font to be used, and the paragraph parameters for the title (in case it is wider than the measure). It also places glue and penalties before and after the title on the galley. It might also add a section number, and record information for the table of contents.

High-level commands are being translated into low-level typesetting instructions. This translation need not be done by LATEX (or indeed by any other TEX macro package). For example, in the `WEB` system of literate programming, much of the work is done using external programs. Similarly, XSLT templates are often used to transform text, prior to it being passed to TEX to typesetting.

For over 10 years the LATEX3 project has been working to enhance LATEX by providing [15, p.1]

> a flexible interface for typographic designers to easily specify the formatting of a class of documents.

Such an interface might, for example, be similar to Cascading Style Sheets (CSS) for HTML. We have seen that Don Knuth only reluctantly added programming features to TEX. The author believes that TEX macros are not a suitable language for creating the above interface, and that the long delay in its delivery is evidence for this.

This interface could instead be written as an external program. In a later section we describe QATEX, which is a wrapper around TEX that effectively allows TEX to interact with external programs.

Improved macros — output routines

This section could also be titled:

```
\let \output = \undefined
```

One of the most interesting and best parts of TEX is the algorithm it uses for breaking a paragraph into lines. The algorithm for breaking the galley into pages is not so good, although for simple technical material it is more than adequate.

In this paper we do not suggest improved output algorithms (we have discussed this elsewhere [4]). Instead, we describe a solution to a related problem. The TEX macro language is not a suitable environment for the writing of complicated page makeup algorithms. Here we describe a means of moving the problem to another domain.

The galley produced by TEX consists of a vertical list. This vertical list consists of boxes, glue, penalties, and so forth (see *The TEXbook*, page 110 for a complete list). The `\showlists` command prints a detailed description of the content of this vertical list.

The output of `\showlists` can parsed by an external program, and used to reconstruct within that program the vertical list created by TEX. If the external program can also send low-level typesetting instructions to TEX, then TEX in effect has become a callable function available to the external program. (As in QATEX, the interactive console or more exactly stdin and stdout can be used for this communication.)

This is not a completely new idea. In 1996, Jiří Veselý asked [10, pp620–621]:

> Once I was asked about the possibility to make a list of all hyphenated words in the book. I was not able to find a way in your book to do this.

To this, Don replied (loc. cit.):

> This would be easy to do now in a module specially written for TEX. I would say that right now, in fact, you could get almost exactly what you want by writing a filter that says to TEX "Turn on all the tracing options that cause it to list the page contents." Then a little filter program would take the trace information through a UNIX pipe and it would give you the hypenated words. It would take an afternoon to write this program; well, maybe two afternoons ... and a morning.

We develop this idea later in the paper.

Instant Preview and TEX as a daemon

This section could also be titled:

```
\let \end = \undefined
```

Interactive programs typically require a response time of less than a tenth of a second, while a response in a hundreth of a second is seen as instantaneous.

On my current 800 MHz PC, the command

```
$ tex story \\end
```

takes about 0.137 seconds, while

```
$ tex \\end
```

takes 0.133 seconds. The first command typesets a small page of material; the second does nothing but start TEX and then exit. Thus, typesetting the small page takes about 0.004 seconds.

It follows from this that typesetting material for Instant Preview is tolerable if the start-up time is included, while it will be perceived as instantaneous if TEX is run as a daemon.

Running TEX as a daemon is an example of TEX forever. We wish for TEX to start up when the computer boots, and to remain running indefinitely. Moreover, we might prefer that there were only a single instance of the TEX daemon running.

Documents and macro packages may have to be adapted, to make the most of Instant Preview. The key concept seem to be this: That the source file be partitioned into regions by markers, which we call 'belays', and that the macro package be able to typeset each region independently. In other words, that the macro package support random access typesetting. This is, again, an example of TEX being enhanced by an improved macro package on the front end.

The author has already written [5] about Instant Preview. At the conference he hopes to demonstrate the latest progress.

Decorating `dvi` files

TEX has no built-in notion of colour, or of graphics inclusion. However, the `\special` command allows device drivers to produce special effects. By decoration we mean the application of colour, change bars and the like to the rendered page.

In the domestic setting, decoration of a room or a house does not move the walls or make other structural alterations. In typesetting, adding decoration should not affect typesetting decisions, such as the line breaks and the placement of items on the page. (This is not to say that the typographic design should not take into account the subsequent application of decoration.)

The current practice regarding decoration is to use a fairly simple `dvi` processor, and to have the (LA)TEX macro package place appropriate `\special` commands into the `dvi` file. From the point of view of a device-driving `dvi` processor, this is probably correct. It seems that, historically, low-level capabilities were added to device drivers. Then macro packages were written to access these new features.

From the point of view of the macro package, this approach is probably wrong. As already noted, decoration should not affect typesetting decisions. This is an important property, whose fulfilment should be central to the approach taken.

Suppose, for example, that some text is to be printed in a spot colour. Placing a `\special` at the start or end of a word does not affect its hyphenation. Therefore something like

```
% usage: \color{red}{Text to go in red}
\def\color#1#2{%
    \special{push #1}%
    #2%
    \special{pop}%
}
```

will suffice, at least in the simplest cases.

However, a page boundary might break the red text. This places a burden on the `dvi` processor, to keep track of this information. Typical `dvi` processors allow random access to the pages in the `dvi` file. Having to look at previous page(s) breaks this random access.

The solution we suggest is to write a `dvi`-to-`dvi` filter that resolves these random access problems. Such a filter is not, of course, a device driver, but it can be used with any device driver. Its purpose is to translate high-level specials into low-level specials.

TEX is being held back by the weakness of tools for decorating text. For example, a common requirement is to place a background rectangle behind a paragraph of text. If the paragraph is broken over two pages, the background rectangle should be similarly broken.

In 1987 Don Knuth and Pierre MacKay discussed a similar problem, namely implementing bi-directional typesetting without extending TEX. They wrote [14, p159]:

> How can we get TEX to do this? The best approach is probably to extend the driver programs that produce printed output from the `dvi` files that TEX writes, instead of trying to do tricky things with TEX macros.

In the same article [13, p161] they then produced a "much more reliable and robust scheme by building a specially extended version of TEX".

QATEX — or ask a friend a question

TEX is a typesetting language, with limited text-processing and other capabilities. Things that are easy to do in other languages are hard to do in TEX. Examples are to find the dimensions of an EPS or other graphics file, or to calculate the sine and cosine of an angle, so that space can be left for rotated text.

Traditionally, such questions have been answered by writing TEX macros. The author finds that TEX macros are not a suitable language for such text manipulation tasks.

Here is an extract from the `\Gread@eps` macro in the LATEX file `graphics.sty`.

```
\immediate\openin\@inputcheck#1 %
\ifeof\@inputcheck
  \@latex@error{File ‘#1’ not found}%
    \@ehc
\else
   \Gread@true
   \let\@tempb\Gread@false
   \loop
     \read\@inputcheck to\@tempa
     \ifeof\@inputcheck
       \Gread@false
     \else
       \expandafter\Gread@find@bb
         \@tempa:.\\%
     \fi
   \ifGread@
   \repeat
   \immediate\closein\@inputcheck
\fi
```

The author has developed QATEX, which allows the TEX macro programmer to ask another process to answer questions. Such as: What is the bounding box of `myfile.eps`?

QATEX (pronounced ‘kwa-tech’) provides a different route. Questions and answers are the essence of QATEX. When QATEX is used, lines such as:

```
!Q=qatexlib.eps.bbox(myfile.eps)
!A=0,0,0 0 35 97
```

appear in the TEX's `log` file.

The first line is a question, produced using a `\write` command. The second line is the answer.

The characters `!A=` are a prompt, produced using a `\message` command.

The remainder of that line is the answer to the question. The prefix `0,0,` tells TEX that the question was successfully posed and answered. There follows the information asked for. This information is supplied by a process external to TEX.

TEX uses `\read -1 to \temp` to read this information from its stdin stream. So far as TEX is concerned, this data might have come from the user. In fact, it has come from a program, namely QATEX.

QATEX works as follows. It is a wrapper progam that invokes TEX, and takes control of its standard input and output. When it sees a question line, followed by the answer prompt, it parses and answers the question, then sends the answer to TEX, using TEX's standard input. However, QATEX does not answer the question itself. It imports a module — in the example above the `eps` module — to answer the question for it.

Here is the definition, in Python, of a function that returns the bounding box of an EPS file, as a string. If not found, it raises an exception. It took me less than 10 minutes to write. It would be part of a `eps` module for use with QATEX.

```
# File: qatexlib/eps.py
_bb_prefix = '%%BoundingBox: '
def qa_bbox(filename):
  f = open(filename)
  for line in f:
    if line.startswith(_bb_prefix):
      sizes = line.split()[1:]
      return ' '.join(sizes)
  msg = "File '%s' has no bounding box"
  raise Exception, msg % filename
```

Alternatives and complements to QATEX

In this section we compare QATEX to shell escape, `eval4tex`, PerlTEX, `sTeXme` and Pymacs. Each of these has some similarity with QATEX.

Shell escape. Modern implementations allow TEX to issue shell commands, as if they had been typed at a command prompt. This allows, for example, a command such as `makeindex` to be run after TEX has processed the body of a document, but before setting the back matter.

However, it also allows other commands to be run, such as the deletion of files. And TEX documents can execute arbitary TEX commands. (Strictly speaking, this is not true. For example, in Active TEX [3] all characters are active. This allows a macro package to prevent execution directly from a document of all but specified commands.)

Often, TEX documents are distributed in source form. If shell escape is enabled, the typesetting a document could result is a shell escape command being run, that finds and deletes all your files. Clearly, shell escape is a security risk. For this reason, shell escape is disabled by default, and is rarely used.

Even without shell escape, TEX macros and therefore documents can overwrite existing files. (TEX has no inbuilt ability to delete files. But it can destroy their contents.) Therefore, modern implementations of TEX refuse to open for writing files that are not in or beneath the current directory, or a similarly specified area.

This restriction is not applied to the reading of files. Therefore, it is possible for a TEX document, when typeset, to include in it other files. These other files might be confidential.

Therefore, in line with the theme of TEX being "just a typesetting system", it might be sensible to:

```
\let \openout = \undefined
\let \openin = \undefined
\let \input = \undefined
```

and have another program take responsibility for security. The security monitor could then send material to be typeset to TEX's standard input stream.

eval4TEX (by Dorai Sitaram) is a two-pass process that allows TEX to send expressions to Scheme for evaluation [1]. It provides a `\eval` macro, that is used as below. (The example is Sitaram's, and my exposition follows his).

```
\eval{(display (acos -1))}  % gives pi
```

On the first pass, the Scheme code

```
(display (acos -1))
```

is written to an auxiliary file, together with some indexing information.

Before the second pass, a helper program `eval4tex` runs Scheme on the auxiliary file, to create a second auxiliary file. On the second pass, the `\eval` command picks up values from the second auxiliary file, and refreshes the data in the first.

As Sitaram writes:

> This strategy is quite common in the TEX world. The popular TEX-support programs BibTeX and MakeIndex, which generate bibliographies and indices respectively, both operate this way.

sTeXme (by Oleg Paraschenko) is another approach to integrating TEX with Scheme [19]. Here is his summary of the goals of the project.

> The (LA)TEX macro language was a great development when it appeared, but now it is

> too out-of-date. Programming in TEX is a fun, but more often it is a pain.
>
> As it seems for me, only very few people can write (LA)TEX macros, but a lot of people would like doing it (like me, for example). This is the problem.
>
> One of the solutions is to provide another scripting language for TEX. That's what is the goal of the sTeXme project. It should provide the Scheme programming language as a TEX scripting language.

This project has two parts, namely an extension to TEX, that allows it to interpret Scheme code, and an extension to Scheme that allow it access TEX internals. We say more on this later.

PerlTEX (by Scott Pakin) uses standard Perl and TEX without extensions [17]. Here is his summary of the goals of the project.

> TEX is a professional-quality typesetting system. However, its programming language is rather hard to use for anything but the most simple forms of text substitution. Even LATEX, the most popular macro package for TEX, does little to simplify TEX programming.
>
> Perl is a general-purpose programming language whose forte is in text manipulation. However, it has no support whatsoever for typesetting.
>
> PerlTEX's goal is to bridge these two worlds. It enables the construction of documents that are primarily LATEX-based but contain a modicum of Perl. PerlTEX seamlessly integrates Perl code into a LATEX document, enabling the user to define macros whose bodies consist of Perl code instead of TEX and LATEX code.

Here is Scott Pakin's equivalent to `\eval`:

```
\perlnewcommand{\reversewords}[1]
  {join " ", reverse split " ", $_[0]}
\reversewords{TeX forever!}
```

PerlTEX, like QATEX, invokes TEX under the control of a separate process. Unlike QATEX, it does not take control of TEX's standard input and output. Invoking `\reversewords` causes TEX to write material to a specially named file. This file corresponds to the question in QATEX. The controlling Perl process then computes the answer to the question, and writes it to another specially named file. Meanwhile, the TEX process goes into a loop, to poll for the existence of the answer file. Once found, it is `\input` by TEX.

PerlTEX seems to have a performance problem. On my 800 Mhz PC, the following example:

```
\documentclass{article}
\usepackage{perltex}
\perlnewcommand{\nothing}{}

\begin{document}
% I've got plenty of nothing ...
\nothing\nothing\nothing
\nothing\nothing\nothing
\nothing\nothing\nothing
\nothing           % 10 nothings
% We're busy doing nothing ...
\end{document}
```

takes about 3.0 seconds to execute. This includes startup time. (On the same machine, it takes QATEX about 1/3000 seconds to do nothing once.)

Here is at least a partial explanation. Instrumenting the code for PerlTEX shows that in compiling the above document, TEX polls for the existence of the helper file approximately 5,000 times. The exact number varies. Adding:

```
\input nothing  % input an empty file
```

to the polling loop *reduces* the time taken to about 1.8 seconds, and reduces the number of pollings to about 500. The UNIX `nice` command could also help here.

Pymacs (by François Pinard) is not a way of using Python with TEX. It is a way of using Python with the Emacs editor [18]. To quote its author:

> Pymacs is a powerful tool which, once started from Emacs, allows both-way communication between Emacs Lisp and Python. Pymacs aims Python as an extension language for Emacs rather than the other way around, and this asymmetry is reflected in some design choices. Within Emacs Lisp code, one may load and use Python modules. Python functions may themselves use Emacs services, and handle Emacs Lisp objects kept in Emacs Lisp space.

Pymacs is mentioned because is was higly influential on the author's approach to the integration of TEX with a scripting language. (At that time, Python had not been chosen.)

Different approaches compared

In the previous two sections we looked at QATEX, shell escape, `eval4tex`, `sTeXme` and PerlTEX. In this section we make some comparisions.

Philosophy QATEX is like shell escape, in that simple queries are sent to another process. The other approaches assume that complex code will be written within, or otherwise produced by, TEX macros. This code is then evaluated by another program.

For example, in QATEX the problem of reversing words might result in the following conversation between TEX and the external process:

```
!Q=qatexlib.string.reverse(TeX forever!)
!A=0,0,forever! TeX
```

The question sent to QATEX could not be along the lines of: "What is the result of evaluating this complex Perl or Scheme expression?" However, such is not the expected use. Rather, it is expected that TEX will send a short and simple query. If the answer is long, it could be placed in an external file. Once that is done, TEX can be told, as the answer, that the file is ready to be `\input` (assuming `\input` is still defined).

Architecture The architecture of the implementation of PerlTEX is closer to that of `eval4tex` than that of `sTeXme`. Perl code is placed in the body of TEX macros, and this code is sent out to Perl for evaluation. Unlike `sTeXme`, and like `eval4tex`, it does not require either an extension to TEX or a special version of the command interpreter for the extension language.

PerlTEX is similar to QATEX in that TEX is run under the control of an external program. However, QATEX uses standard input and output for communication, whereas PerlTEX polls named files.

QATEX provides an efficient and portable low-level interface between TEX and an external process. PerlTEX is a higher-level package. There is no reason why the QATEX interface, or something similar to it, should not be used by PerlTEX, so as to improve performance. The same applies to `eval4tex`, where using this interface would remove the need for a second run. It would also provide better interaction.

Security Any process that evaluates code supplied by a document will expose a security problem, unless the code evaluator is already secure (as in Java, for example). PerlTEX provides security by using a secure Perl sandbox. QATEX provides security by having TEX (and thus the document) supply only data.

Of course, any program that evaluates untrusted data as if it were trusted code has a security issue. If it is necessary to evaluate safely untrusted code, then a secure sandbox is required. This is the key security issue. QATEX is a small interface application, which does not attempt to solve this problem. There is no reason why QATEX should not be used with such a secure sandbox. But that is a matter for the developer who builds upon QATEX.

Integration and extension Of all the projects considered in this section, `sTeXme` is the most ambitious. It involves making major extensions to TEX, to produce a new program, called `sTeXme`.

The new name is necessary. TEX experts will already know that Don Knuth does not object to the sources of TEX the program being used as the basis for creating a superior program. However, he is most insistent that programs that are not TEX must not be called TEX. More exactly, in [8, p572] he wrote:

> That is all I ask, after devoting a substantial portion of my life to the creation of these systems and making them available to everybody in the world. I sincerely hope that the members of TUG will help me to enforce these wishes, by putting severe pressure on any person or group who produces any incompatible system and calls it TEX or METAFONT or Computer Modern — no matter how slight the incompatibility might seem.

This insistence on the stability and consistency of TEX is, in this author's view, a significant contribution to its longevity. Users know what to expect, and get what they expect, when they use TEX.

Regarding the scope of his new program `sTeXme`, Oleg Paraschenko reports:

> [...] Scheme code can be executed from TeX and that Scheme code can access TeX internals (getting a string from the TeX string pool, getting a macro definition as the Scheme list).

The source file `stexmelib.c` on the SourceForge repository indicates that Paraschenko is building a C-coded Scheme extension, in which equivalents to TEX boxes and the like can be stored. This indicates that there are many points of contact between his project and the next section.

PyTEX

The goal of the PyTEX project is to combine Python programming with TEX typesetting. To understand this, think of Tcl/Tk: Tcl is a scripting language and Tk is a toolkit for building GUI programs. Perl and Python also have interfaces to Tk, allowing them to use Tk when building GUI programs.

Now think of LATEX as LA/TEX. LA is a front end to the TEX typesetting program written in TEX's macro language. PyTEX, or Py/TEX if you prefer, is to be a front end to TEX written in Python.

PyTEX replaces the part of TEX that Don Knuth said he did not want to write, namely the TEX macro programming language, with something more widely used. Our aim is to provide an alternative means of programming typesetting decisions and logic. This will make TEX easier to use.

Here is an example of the interface. We are in Python, and wish to typeset a paragraph of text, namely

The cat sat on the mat.

to a measure of 6 picas, which is about one inch (or 25 millimetres). This is a toy example. After typesetting the paragraph, we wish to bring it into Python for page makeup.

To see how this is done, issue the command

```
$ cat cat_sat.tex | tex | grep '^[.\]'
```

where the source stream is

```
% cat_sat.tex
\tracingonline 1
\showboxbreadth \maxdimen
\showboxdepth \maxdimen
\scrollmode

\setbox0\vbox{%
\hsize  6pc
The cat sat on the mat.\par
\showlists
}
```

The annotated output of the grep is:

```
# This is an annotation.
# Start of the first line of paragraph.
\hbox(6.94444+0.0)x72.0, glue set 0.58331
# indentation box, 20pt wide
.\hbox(0.0+0.0)x20.0
# The word "The".
.\tenrm T
.\tenrm h
.\tenrm e
# normal interword glue
.\glue 3.33333 plus 1.66666 minus 1.11111
# The word "cat".
.\tenrm c
.\tenrm a
.\tenrm t
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm s
.\tenrm a
.\tenrm t
# Zero width line filling glue.
.\glue(\rightskip) 0.0
# Penalty for breaking page at this point.
\penalty 300
# Interline glue.
\glue(\baselineskip) 5.05556
# Start of second line of paragraph.
\hbox(6.94444+0.0)x72.0, glue set 20.88878fil
.\tenrm o
.\tenrm n
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm t
.\tenrm h
.\tenrm e
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm m
.\tenrm a
.\tenrm t
.\tenrm .
# Inserted by TeX, for internal reasons.
.\penalty 10000
# Allow the last line of para to be short.
.\glue(\parfillskip) 0.0 plus 1.0fil
.\glue(\rightskip) 0.0
```

This is a complete description of the paragraph formed by TEX's line breaking algorithm.

This is the essence of the interface between Python and TEX. Material is sent to TEX to be typeset, say using stdin. The `\showlists` command is used to write the results to stdout, from which they are picked up by Python.

On the input side, Python is responsible for parsing the input stream, and placing appropriate items on the horizontal list. It is also responsible for ensuring that nothing inappropriate is placed on the list. The whole process should not generate TEX errors (although warnings about overfull boxes and so forth are welcome).

On the output side, Python parses the output stream, and from it reconstitutes the boxes that TEX has formed, thus forming a Python object.

In Python code, our example might look like

```
hlist = Hlist()  % new horizontal list
text = "The cat sat on the mat."
hlist.extend(text)
vlist = hlist.linebreak(hsize=6*pica)
```

where hidden in the call to the `linebreak()` method there is a sending of data to TEX, and a reconstruction in Python of the boxes that TEX built. From here on, the output or page-makeup routine can be written in Python. Note that `cat_sat.tex` invokes no TEX macros.

Conclusions

In the 15 years since TEX has been frozen, very few deficiencies have been found in the algorithms it uses for breaking a paragraph into lines, for typesetting mathematics, and for setting tables. Since 1990 there has been little (but valuable) progress in the area of Unicode fonts, for which an extension of TEX genuinely is needed. TEX was written to be archival, and it is holding up well after its first quarter-century or so.

There are many problems in our use of TEX. This paper has discussed several:

- coloured text and other decorations,
- interactive use of TEX,
- input transformation,
- programming page makeup.

All of these arise not out of TEX itself, but out of the way in which we use TEX.

There is an irony in our use of TEX macros. Recall that when Don was looking at the design of TEX he found that: [11, p.648]:

> Every system I looked at had its own universal Turing machine built into it somehow, and everybody's machine was a little different from everybody else's.

He then went on to say:

> I was tired of having to learn yet another almost-the-same programming language for every system I looked at; I was going to try to avoid that.

What we have now with TEX macros is a Turing machine very different from any other. This is just what he wished to avoid. However, QATEX provides a powerful complement to existing TEX macro packages, and PyTEX will use TEX as "just a typesetting language", which is what Don wanted it to be in the first place.

In 1996, Piet van Oostrum asked Don Knuth about TEX's macro programming language [11, p648–9]:

> I don't know if you have ever looked into the LATEX code inside, but if you look into that, you get the impression that TEX is not the most appropriate programming language to design such a large system. Did you ever think of TEX being used to program such large systems and if not, would you think of giving it a better programming language?

In response to this, Don Knuth said (loc. cit.):

> It would be nice if there were a well-understood standard for an interpretive programming language inside an arbitary application. Take regular expressions — I define UNIX as "30 definitions of regular expressions living under one roof." [*laughter*] Every part of UNIX has a slightly different regular expression. Now, if there were a universal simple interpretive language that was common to other systems, naturally I would have latched onto that right away.

The theme of this paper is TEX typesetting with fewer macros. We use instead a "simple interpretive language", namely Python. If we learn to use TEX in new ways, and take good care of it, TEX will be good for its second quarter-century.

Long live TEX!

References

[1] eval4tex, `http://www.ccs.neu.edu/home/dorai/tex2page/eval4tex-doc.html`

[2] Jonathan Fine, Editing `.dvi` files, or Visual TEX, *TUGboat*, **17** (1996), 255–259

[3] ———, Active TEX and the DOT input syntax, *TUGboat*, **20** (1999), 248–254

[4] ———, Line breaking and page breaking, *TUGboat*, **21** (2000), 210–221

[5] ———, Instant Preview and the TEX daemon, *EuroTEX 2001 Conference Proceedings*, 49–58

[6] ———, TEX as a callable function, *EuroTEX 2002 Conference Proceedings*, 26–35

[7] Donald E. Knuth, The Errors of TEX, *Software — Practice and Experience*, **19** (1989), 607–685. (Reprinted in [9])

[8] ———, The Future of TEX and METAFONT, *TUGboat*, **11** (1990), 489 (Reprinted in [13])

[9] ———, *Literate Programming*, CSLI (1992)

[10] ———, Questions and Answers II, *TUGboat*, bf 17 (1996), 355-367 (Reprinted in [13])

[11] ———, Questions and Answers III, *MAPS (Minutes and APpendiceS)*, **16** 1996, 38–49 (Reprinted in [13])

[12] ———, The future of TEX and METAFONT, *TUGboat*, **11** (1990), 489 (reprinted in [13])

[13] ———, *Digital Typography*, CSLI (1999)

[14] Donald E. Knuth & Pierre MacKay, Mixing Right-to-Left Texts with Left-to-Right Texts, *TUGboat*, **8**, (1987), 14–25. (Reprinted in [13])

[15] Frank Mittelbach & Chris Rowley, The LATEX3 Project, `http://www.latex-project.org/guides/ltx3info.pdf`

[16] NTG TEX future working group, TEX in 2003: Part I Propositions and Conjectures on the Future of TEX, *MAPS (Minutes and APpendiceS)*, **21** 1998, 13–19

[17] PerlTEX, `http://www.ctan.org/tex-archive/macros/latex/contrib/perltex`

[18] Pymacs, `http://pymacs.progiciels-bpi.ca`

[19] sTeXme, `http://stexme.sourceforge.net`

The TEI/TEX Interface

Sebastian Rahtz

February 28, 2005

Abstract

In the view of many people, the natural way to prepare a typeset document is to use LATEX or ConTEXt. It produces high-quality output, the source document is a clean mixture of text and markup, and it works on any computer. For another group of people, however, the natural way to proceed is to prepare a validated XML document which can be used to either make a web page or to make a printed document. One choice of an XML encoding for this group is the Text Encoding Initiative (TEI) scheme.

This paper is in two parts. The first part examines the arguments for and against authoring in XML, rather than TEX, and demonstrates how some common TEX situations are catered for in TEI XML.

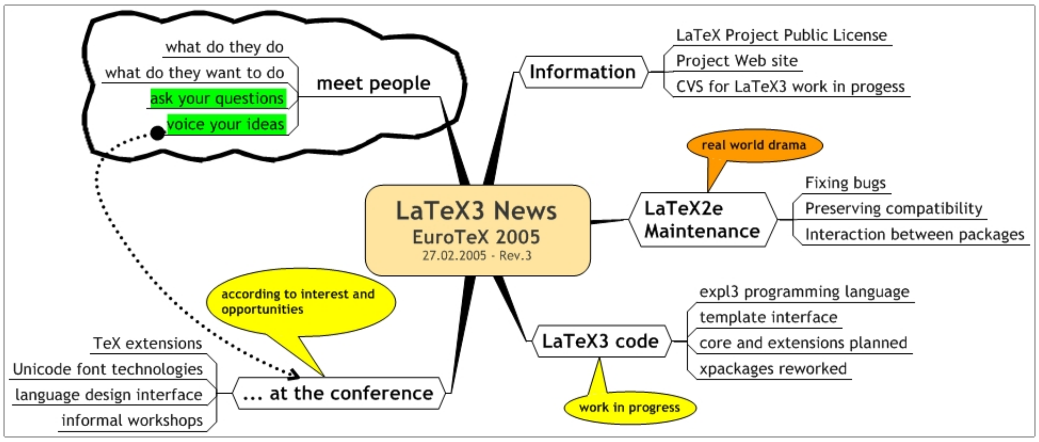
The second part of the paper examines how, if we *do* choose XML, we can continue to harness the power of TEX. We examine the four main routes of

a) using a modified TEX to read XML directly;
b) translating XML direct to high-level TEX;
c) translating our XML to another XML which is functionally identical to LATEX and then translating that; and
d) translating XML to an XML-based page description language (XSL FO), and processing that with TEX.

None of these is completely satisfactory, and we end by considering what hope there is for the future.

LATEX3 News

Frank Mittelbach Chris Rowley



The main purpose of this event is to introduce the human side of the LATEX3 Team and to explain what we are doing, both collectively and individually, to support and promote LATEX and automated typography.

Here is a summary of the team's recent, current and planned activities.

- General News
  - LATEX Project Public License
  - The project Web site
- Maintenance
  - Fixing standard LATEX
  - Helping packages work better with each other
- LATEX3 Code — work in progress
  - Web access to experimental LATEX3 code at `http://www.latex-project.org/cgi-bin/cvsweb.cgi/`
  - The next version of the LATEX3 Programming Language, `expl3` at `http://www.latex-project.org/cgi-bin/cvsweb.cgi/experimental/expl3/`
  - Progress on `xpackages` (e.g., `xor` learns to balance) at `http://www.latex-project.org/cgi-bin/cvsweb.cgi/experimental/xpackages/`
  - Plan to provide full core and typical extensions based on `expl3` and the `template` mechanism
- Work at this conference
  - Investigate TEX extensions and quasi-TEX 'extensions' and their consequences for LATEX
  - Investigate 'Unicode font technologies' (XeTEX and friends) possibly including 'Unicode-encoded math chars/glyphs'
  - Work on language interface design

There will be plenty of opportunity for questions and discussion of our plans.

The 16 Faces of a Dutch Math Journal

Hans Hagen

Abstract

Much of what ConTEXt originally provided originated from our daily needs, at that time dictated by educational consultancy and course development. However, the last couple of years most features find their origin in the demands of publishers, users as well as an occasional "Let's see (prove) if TeX can do it (better)". One of those users is the Dutch Math Society (NAW).

Quite a while ago the Dutch Math Society decided to restyle their journal and the decision was made to use ConTEXt as typesetting engine, one reason being its ability to typeset on a grid and place graphic in columns. Since it happened in the early days of ConTEXt, some of the demands resulted in rather complex and often weird macros.[1]

Advanced mixed font support, magazine-like column features, tight integration with MetaPost, flexible placement of elements etc. are nowadays supported in the kernel in a more natural way, if only because the core of ConTEXt has become more flexible and mature. And so the moment has come to let the editors switch to the reimplemented journal style.

In this talk I will illustrate how needs by demanding users like the Dutch Math Society's Journal have brought ConTEXt to where it stands today and is heading tomorrow.

[1]These were written by Taco Hoekwater, who did a pretty good job, as proven by the fact that up to date these macros are still in use.

Typographic Perfection with OpenType?

Adam Twardoch

February 26, 2005

Abstract

In September 1999, Adobe Systems declared their PostScript Type 1 font format "obsolete". Until then, this font format was dominating the professional pre-press and printing business, but now was to be replaced with OpenType – a font format developed by Microsoft and Adobe, with collaboration from Apple. Four and a half years later, OpenType is a fact: both the world's largest font foundries and individual type designer publish new fonts in this format.

OpenType fonts have numerous advantages: they can be used in many operating systems without any conversions (Windows 9x/2000/XP, MacOS 9/X, some Unix environments); they use the universal character encoding standard Unicode; finally, they can include typographic layout features that allow for comfortable use of ligatures, small caps, swash alternates or old-style numerals, as well as more advanced functionality such as justification alternates.

You may have heard that Unicode is the only solution for the encoding mess in electronic text processing. You may have also heard that OpenType is the new cross-platform font format that enables unprecedented typographic perfection. Adam Twardoch will present these technologies and discuss how much truth and how much myth these promises hold.

Bio:

Born 1975 in Poland, Adam now lives in Frankfurt (Oder), at the German-Polish border. He is Scripting Products and Marketing Manager at Fontlab Ltd., an international software vendor specializing in font editors and typography products. He serves as typographic consultant to MyFonts, a major online font distributor. Adam provides consulting services in font creation, font tool development, font technology and multilingual typography for Adobe, Bitstream, Corel, Linotype, Microsoft and other clients. Adam regularly writes and lectures about fonts and typography. He is member of Association Typographique Internationale (ATypI) and of the Polish TEX Users' Group (GUST).

Namespaces for εχTEX

Gerd Neugebauer

October 31, 2004

Namespaces for TEX are a long awaited extension. In this talk the requirements for such an extension are described. Namespaces primarily restrict the visibility of macros and active characters. Thus the probability of name clashes is reduced. As addition one can imagine to apply namespaces to other entities like registers, catcodes etc as well.

εχTEX is an attempt to reimplement TEX. The major goals behind the reimplemenbtation are a modular and configurable structure tailored towards experiments and extensibility.

Fortunately the integration of namespaces can be located at very few places in the εχTEX architecture. As a consequence an implementation idea for εχTEX can be sketched and the experimental implementation in εχTEX is shown.

1 Introduction

With the vast amount of packages emerging on CTAN the need came up to separate the packages. In other programming languages this is accomplished by using namespaces – sometimes also called modules or packages. The classical TEX system lacks such a mechanism.

In this proposal an analysis is provided which shows where modifications of a TEX-like system are necessary to implement namespaces. The proposed solution tries to be minimalistic. This means that one goal is to change the underlying system as few as possible. In addition existing TEX code should continue to work without any change.

2 Encapsulation

The primary requirement for namespaces is that they encapsulate the internals of "modules". This means that each information about the state is hidden in the outside world. The access is enabled via well-defined interfaces only.

In this document the focus is put onto namespaces for control sequences and active characters.

3 Backward Compatibility and Initialization

The namespace extension should be backward compatible. This means that the behaviour of the system should not depend of the use of a namespace. This can be achieved by using a default namespace in any case where no other namespace is specified.

On the other hand the definitions in the namespace should be properly initiated. Thus we want to ensure that the namespaces can be used without the burden of too much initialization.

4 Definition of Namespaces

The definition of the current namespace is just a string to be kept somewhere. In terms of TEX it is advisable to store the current namespace as tokens in a special tokens register to allow read and write access to it.

Consider the name `\namespace` for this toks register then the following instruction can be used to advice TEX to set the appropriate namespace:

```
\namespace{tex.latex.dtk}
```

The default namespace should be denoted by the empty toks register:

```
\namespace{}
```

The wellknown primitives `\the` and `\showthe` can be used to get access to the current value of the namespace:

```
\namespace{tex.latex.dtk}
\the\namespace
```

5 Communication between Namespaces

Namespaces provide a means for separation of different modules. Thus we need a way to communicate between namespaces. For this purpose we want to provide a primitive to declare that a certain set of entities are visible from the outside. All entities not declared to be visible are purely private.

The primary entities to consider are macros and active characters. They are characterized by tokens. Thus we can again use a token list to keep this information.

Consider the name `\export` for the names toks register. Then the following example declares that the macros `\abc` and `\xyz` and the (active) character `~` can potentially be accessed non-locally:

```
\export{\abc \xyz ~}
```

The other side of communication is the import of the exported entities into another namespace. In analogy to the name export we want to call this functionality import. For the import it is sufficient to name the namespace to be imported:

```
\import{tex.latex.dtk}
```

The semantics is that all entities exported by the namespace – i.e. contained in the toks register `\export` – are assigned to in the current namespace as well. This is similar to a multitude of let invocations.

As a consequence of this definition the meaning of a macro can be changed in the defining namespace without affecting the meaning in the importing namespace.

6 Namespaces and Groups

Since the main task of the declarations if performed by special tokens registers it is clear that the namespaces are coherent with the groups structure already present in TEX.

In the following example the macro `a` is defined in the namespace `tex`. Since the group ends afterwards the namespace returns to its previous value. Thus the definition of `\y` is performed in the outer namespace.

```
\begingroup
  \namespace{tex}
  \gdef\x{123}
\endgroup
\def\y{123}
```

One question which arises is, how the macro `\import` interacts with the grouping. The answer to this question is that the import should influence the current group only. Similar to the definition of `\let` the prefix command `\global` can be used to indicate that the imports should be applied globally instead of locally in the current group:

```
\global\import{tex.latex.dtk}
```

The macro `\import` has to take into account the `\global` prefix.

Another inference of namespaces and groups can be seen in the following example:

```
\begingroup
  \namespace{one}
  \global\export{\x}
  \gdef\x#1{-#1-}
\endgroup
```

Note that the `\export` declaration is preceded by a `\global` modifier. Consider the case that this modifier would not be there. Then the end of the group would revert the meaning of `\export` the previous binding. In general this would destroy the intended meaning. The `\general` modifier ensures that the intended meaning of `\export` survives the end of the group and can be used in subsequent imports.

7 Namespaces and Expansion

Let us consider the following example where a macro with an argument is exported:

```
\namespace{one}
\begingroup
  \namespace{two}
  \global\export{\x}
  \gdef\x#1{-#1 \y-}
  \gdef\y{in one}
\endgroup
\import{one}
\def\y{two}
\x\y
```

The intuitive meaning of the last expression `\x\y` is that `\y` is taken from the namespace `two` and `\y` is taken from the outer namespace `one`. Now we follow the expansion of `\y`. It leads to

```
-\y␣\y-
```

where the first `\y` comes from the argument. As such it is rooted in the outer namespace `one`. Whereas the second `\y` is defined in the namespace `two`.

As we can conclude that the namespace has to be attached to the token. Then the two tokens can be expanded in their original namespaces. To illustrate this we use the namespace as subscript to the macro name. With this convention we have the following tokens in the situation above:

-\y$_{one}$␣\y$_{two}$-

With this refined understanding we can come to the conclusion that the expansion will indeed lead to the expected result:

```
-two in one-
```

8 Explicit Expansion without Import

In several programming languages there is the possibility to invoke a method in another namespace. With the means we have depicted so far we can achieve the same effect:

```
\begingroup\namespace{tex}\expandafter\abc\endgroup
```

This construct can even be used to expand a macro which is not exported. Since it is in general not a good idea to use this feature no provision is made to provide a shortcut for such an expansion.

In this document a minimalistic approach has been proposed. To provide some syntactic sugar would require some more adaptions to the basic machinery. Since all basic requirements can be fulfilled within the minimalistic approach it has net been considered worthwhile to explore these adaptions any further.

9 Namespaces and the Basic Definitions

Usually nobody actually starts with iniTEX since nearly no definitions and settings are defined in it. At least something like the `plain` definitions are used. With the means given in the previous sections each new namespace would start out like a new iniTEX instance.

To solve this problem, we define a fallback strategy for the resolution of control sequences and active characters. If a definition is not found in the current namespace then the definition is taken from the default namespace.

The default namespace is denoted by the empty token list. Initially the namespace is set to the default namespace.

With this definition it is possible to load the some macros into the default namespace – e.g. the `plain` definitions. Then they are availlable in any namespace unless redefined in it.

10 Implementation in ε_χTEX

ε_χTEX (`http://www.extex.org`) is a project to provide an implementation of a typesetting system based on the ideas of TEX. It is designed to be highly configurable and should provide a base for extensions and experimentation. As a starting point a TEX compatibility mode is provided.

According to the considerations in the previous sections we need the following additions to εχTEX:

- The definition of tokens and their factory have to be extended to carry the namespace.
- The group has to be extended to provide means to store the current namespace.
- The binding mechanisms for control sequences and active characters needs to be extended to take into account the fallback to the default namespace.
- The primitive `\namespace` has to be provided which allows the reading and writing access to the namespace stored in the group.
- The primitive `\export` has to be provided to allow access to a special tokens register in the current namespace.
- The primitive `\import` has to be provided which is similar to the implementation of `\let`. New bindings for the exported control sequences or active characters have to be introduced to reference the definitions in the defining namespace.

11 Namespaces for Registers

In the current extension of εχTEX the registers are not affected by the namespaces. Nevertheless it might be desirable to extend namespaces to registers.

For instance count registers can be made aware of namespaces. Then each namespace can have its own incarnations of count registers. The extensions into this direction is straight forward. Experiments into this directions have been performed within εχTEXNevertheless they where not really convincing. The plain format makes use of several count registers. The adaption of the visibility of count registers would require adaptions on the macro level as well.

The restrictions to a limited number of count registers has already been relaxed in εTEX and in εχTEX even further. Thus with the use of the allocation macros it is no problem to have separate registers in separate modules – even when namespaces are used.

12 Conclusion

In the previous sections we have seen how a basic namespace support has be integrated into εχTEX. The changes required for this extension are restricted to very few modifications in the core components. These modifications provide a base upon which the externally visible extensions can act. The extensions are purely optional – to be enabled in a configuration file or even loaded dynamically within εχTEX. Without the definitions of the three new primitives the behaviour of εχTEX has not been changed.

The base mechanism for the use of namespaces is provided. Now it is up to the macro level to make use of it.

contextgarden.net

Patrick Gundlach

February 25, 2005

Abstract

The goal of the *contextgarden.net* project is to enhance the documentation of ConTEXt. It consists of several web services that together provide the technical framework behind the documentation. A large (and growing) percentage of the supplied content is actually provided by the visitors of the interconnected web sites.

1 Introduction

Many users have tried the LATEX-alternative ConTEXt. But more than a few have given up almost right at the start, simply because they didn't know how to proceed any further. Unnecesarily so, since there is an active user group that is very helpful. And there is a vast amount of information and documentation on ConTEXt readily available. One just has to find it.

2 What is available?

The official documentation for ConTEXt is available on the web server of PRAGMA ADE[1]. There, you will find many PDF-files that can answer your questions as well as show you some interesting possibilities of typesetting with ConTEXt. There are two distinct ways to access the documentation on the web site.

The first possibility is to use PDF-based navigation. Just select *showcase* on the web site of PRAGMA ADE, and you will get a hierarchical overview of all available PDF-documents. The navigation itself shows some aspects of the vast possibilities of ConTEXt. This makes it plain to see that it will be worth having a closer look at the program, even before you have read the first document!

The second possibility is to is to select *overview* on the web page. This will present you with a rather simple listing of the available files based on their category. You can see all available manuals and some supplementary documentation at a glance.

The novice user should definitely have a look at the beginner's manual *ConTEXt, an excursion* and, after that, *ConTEXt, the manual*.

Besides these two important documents, there are different sets of manuals:

manuals This is the most important set of documentation files. Besides the beginner's manual and the main manual, there are manuals dealing with XML processing, grid-based typesetting, stepcharts, MetaFun and MathML, and more.

magazines This relatively new set describes smaller aspects of ConTEXt and typesetting. A specific volume, for example, is about formatting digits, and another volume is about hiding parts of section titles inside running heads.

qrcs quick references. Each one of these documents contains a list of all available user level commands within a language interface. For every command, its general syntax is described along with the list of allowed parameters and keywords, but nothing else.

sources sample documents with source code. Currently this set consists mainly of the presentation styles that are shipped with ConTEXt. The source of these styles is also well documented using TEX comments.

technotes At the moment there is only one article about graphic inclusion and positioning in PDF available.

[1] `http://www.pragma-ade.com`

uptodate like the special documents in the section *manuals*, the *uptodate* documents are about specific issues in ConTEXt. There are manuals about flowcharts, tables, typesetting Chinese and JavaScript. Nowadays most of the *uptodate* documents are renamed and put into the *manuals* section.

3 What is missing?

Considering the great number of manuals, it may sound strange that there could be something missing. But those who look at *the manual* will probably notice that it lacks a chapter about typesetting tables. There is a section about a simple table variant (*tabulate*), but it is not explained in any detail. Another variant (*table*) appears often in the examples, but a more helpful explanation is missing. And it lacks an overview of the many different options that can be used when trying to typeset a table.

Another thing you will not find in the printed manuals are practical hints. Things like installation issues, design discussion, tricky problems that are due to misunderstanding or misconfiguration, et cetera.

Yet another problem is that the descriptions in the manuals are partly outdated. The *uptodate* (or the descendants) are, despite their name, already several years old. And because the development of ConTEXt is quite fast, the descriptions of the commands are often incomplete and in some areas, even the underlying concepts have changed. You can only keep track of these things by keeping an eye on the mailing list and watching the changes in the ConTEXt distribution.

Moreover, because there are so many documents, you can actually lose the grand picture. Where are the rules for grid typesetting? What line separators am I allowed to use in *tabulate?* The answers are spread throughout the documents. A global index would be helpful.

A specific request that frequently arises on the ConTEXt mailing list is the lack of sample documents with source code. On the web pages of PRAGMA ADE you can download a few: the magazines (ThisWay); the pdfTEX manual; and the commented sample presentation styles. But for many users this is not sufficient.

Yet another issue that some users are unhappy about has nothing to do with documentation. ConTEXt is not perfectly supported by the TEX-distributions, although the situation has greatly improved over the last years. "Just" trying out ConTEXt sometimes fails because the local distribution lacks some files or because it contains a way too old version of ConTEXt. At present there is a change in the TEX directory structure in *web2c* and accordingly a change of the directory layout of ConTEXt. This brings a whole new set of compatibility problems.

4 In the garden

With the project *contextgarden.net*, I provide some applications and web services that jointly try to address the problems mentioned above. Right now, the following services are available: the wiki, texshow-web, live ConTEXt, source browser and archive. The services are all linked from the main page.[2] *contextgarden.net* is kindly sponsored by DANTE e.V., the German TEX user group.

4.1 Wiki

A wiki is a service, where the visitors of the web page create content. Every visitor can add, change and delete the web pages. The following definition is taken from the wikipedia, a free encyclopedia that is based on the same principle:[3]

> A Wiki or wiki (pronounced "wicky" or "weekee") is a web site (or other hypertext document collection) that allows a user to add content, as on an Internet forum, but also allows that content to be edited by any other user [...] Wiki wiki comes from the Hawaiian term for "quick" or "super-fast".

On every wiki page there is a button labeled "edit", that lets you modify the page content. The wiki syntax is simple, plain ascii text with some extensions for mark-up, such as = for heading and *, # for items in an unordered and ordered list respectively. The following example should be easy to understand for all LATEX and ConTEXt users.

[2] `http://contextgarden.net`
[3] `http://en.wikipedia.org/wiki/Wiki`

```
== This is the main title ==
Every page should begin with an introductory
paragraph.

* first item in an unordered list
* and the second

<code>
an environment like verbatim or \starttyping ...
\stoptyping
</code>
```

A table of contents is inserted automatically if there are enough headings on a page.

The main application on *contextgarden* is a wiki especially for ConTEXt. Its content is mainly filled by only a few, but quite active, users of the site. There are about 100 pages on the different topics that cannot be (easily) found in the manuals. One of the most obvious differences between the wiki and the manuals is that in the wiki, you *can* find practical texts like installation experiences and overviews such as a list of text editors with ConTEXt support. Every user can upload TEX and PDF files to the site, and therefore the wiki is a good platform for the publication of sample documents.

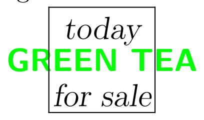
One of the important features in a wiki is that you can get a list of recent changes. This leads to a pragmatic way of editing content: one user creates a page on a specific subject, that is not perfectly worded nor 100% complete yet. Afterwards, other users complete this page by supplying their own knowledge on the subject. As a result, the quality of the page increases over time.

Even news items are put into the wiki. For example, the users can see the list of included changes whenever a new ConTEXt distribution is released. The wiki was started around July 2004. In the long term it should be a full complement to the official manuals.

The wiki on *contextgarden* has several features that makes it well suited for ConTEXt documentation. One extension is the direct rendering of ConTEXt input. Taking the following input:

```
<context>
\defineoverlay
   [tea]
   [{\green \ss \bf GREEN TEA }]
\framed [height=40pt,
         background=tea,
         align=middle]%
    {\em today \blank for sale}
</context>
```

you get



The wiki internally passes the text to ConTEXt to be typeset. This creates a PDF file from the input, and then the wiki calls on ghostscript to convert this PDF file to a PNG image. The end result is that you can view the typeset example directly from within your web browser.

Also included is a pretty printer for TEX and XML source. This formats and colors source code for readability. Perhaps you are familiar with this behaviour from your text editor. The final extension to the wiki is the ability to create hyperlinks to *texshow-web* (see below). With

```
<cmd>adaptlayout</cmd>
```

a link will be created. When that link is clicked, the page that contains the definition of the command *adaptlayout* within *texshow-web* will be opened.

4.2 texshow-web

texshow-web is an alternative implementation of the perl/Tk program *texshow* that comes with every ConTEXt distribution.

What follows is a quick summary for those of you who are unfamiliar with *texshow*: This program gives an overview of all user commands. The full set of the parameters and arguments belonging to a specific command can be shown in a syntactical overview, like this one:

```
\adaptlayout[...,...,...][...,...=...,...]

   [...,...,...]   number
   height          dimension max
   lines           number
```

The output is colored, so that you can easily see what parameters you can use in each argument. In this example the first parameter accepts a list of numbers whereas the second parameter takes a list of assignments. Allowed keywords for the assignment are *height* and *lines*, and the allowed values for height are a *dimension* or the word *max* and for lines a *number* (within TEX's limits of course).

The new web-based variant, *texshow-web* makes it possible to add a comment, a description and any number of examples to the a specific command. Of course *texshow-web* also offers a full-text search, so that you can find the command you need with more ease. As with the wiki, all users have the possibility to complete missing information or to correct pages when needed.

ConTEXt users often struggle because one not only has to know all of the possible parameters for a command by name, but one also has to understand their effects on typesetting. In the manuals the parameters are only partly described. For example, there is no explanation of the parameter *beforehead* in an *itemize* environment. That is why *texshow-web* has a field *description* where this kind of information can be stored. Currently, there are only a few entries with that additional content, but progress is slowly being made.

The original *texshow* program from the distribution has recently been adapted to show all the comments, descriptions and examples from *texshow-web* as well, but does not allow editing.

Features currently in development are: a way of categorizing commands into logical units (graphic inclusion, section and headers, typographic commands), a multi-lingual user interface, and documentation for ConTEXt's programming interface (API). The API documentation will cooperate with the *source browser* (see below) that shows the definition of the commands within ConTEXt's source code.

Until now, *texshow-web* has only been used for ConTEXt documentation. But it should be possible to use it for LATEX documentation without big difficulties.

4.3 Archive

There were already several searchable archives for the ConTEXt mailing list: the NTG has one and the news-mail portal Gmane has one as well. Slavek Zak used to host one, too. There are some other, less popular, archives as well. But none of the those is complete as well as easily searchable. Therefore I have installed yet another mailing list archive at *contextgarden.net*. This one is almost complete, very quick and searchable. The two new lists are also archived on this server: the ConTEXt developer list and the *foxet* (a ConTEXt based XML-FO processor) list.

4.4 Live

Live ConTEXt is an on-line ConTEXt typesetting service. You type your document into a web-form and after you submit the form to the server, your source gets processed by *texexec* and typeset. The resulting pdf document as well *texexec*'s screen output can be viewed online or saved to your harddisk. This makes it possible to use ConTEXt without actually installing it. The underlying TEX system uses the latest teTEX beta and ConTEXt distribution. *Live ConTEXt* is also used as a sort of reference installation. Errors that do occur on a local system but cannot be reproduced on *Live ConTEXt* are most likely a local problem only.

4.5 Source

With the *source browser* you can view ConTEXt's source code. Using a simple navigation system you have access to the almost 600 files from the current distribution. This is especially interesting to the programmers that need to see the definition of the commands. The included full-text search helps finding commands. You can access the definition of the commands via hyperlinks: `http://source.contextgarden.net/tex/context/base/core-pos.tex#setpositions` and `http://source.contextgarden.net/core-pos.tex#setpositions` points to the definition of *setpositions*, without having to know the line number in advance.

5 Future work

The services mentioned here are provided by *contextgarden.net*. The most important one is certainly the *wiki*. Until now it was not necessary to structure the information in the wiki. If it continues to grow as it did in the past, a more formal structure will be necessary. *texshow-web* should become the preferred reference for the ConTEXt commands. The resources provided by *texshow-web* will be used by the existing tools (*texshow*) and documentation in future releases. Because the number of descriptions, comments and examples is still low, the ConTEXt community is asked to fill in the missing information.

Currently a search engine is in development that allows the user to search *texshow-web*, the *wiki*, the manuals and the mailing list archive all from one place. This should provide an even more powerful tool in your quest to find the necessary information.

Experiences with micro-typographic extensions of pdfTEX in practice

Hàn Thế Thành
University of Education,
Ho Chi Minh City,
Vietnam

February 21, 2005

Abstract

pdfTEX provides two micro-typographic extensions: margin kerning (also known as character protrusion) and font expansion. Those extensions have been available for a while, however they are not used much yet, probably due to their complicated setup and not that visible benefits they bring. In this article I want to share some experiences, either good or bad, in using those extensions in practice, the tricky parts of them and how to get the best from what pdfTEX offers without having to know all the low-level details and messy font issues.

1 Introduction

Font expansion and margin kerning have been introduced several times in various articles, so I won't go to any detailed description here. From a practical point of view, what those extensions bring is pretty simple:

1. Margin kerning makes the margins of text *look* smooth, by moving certain characters out to the margins by a small amount. The most common case is to move the hyphen character or punctuation marks, but applying this technique to certain letters also improves the result.

2. Font expansion can help to improve line-breaking. Typically, a text typeset using font expansion has:

   (a) less hyphenations,

   (b) less overfull and underfull boxes,

   (c) more equal inter-word spacing ,

   (d) reduced occurrence of "rivers".

   These benefits are most visible in difficult cases, like narrow-column typesetting, disabled hyphenation, or simply in automated typesetting when manual work needed to correct problematic cases should be minimized or totally avoided.

Margin kerning and font expansion have been implemented in the *hz* program by Hermann Zapf and URW; that's why I called them together *hz* extensions of pdfTEX[1]. For more detailed information on margin kerning and font expansion, including background and related works on *hz* extensions in other systems, please refer to [Thành 2001]. From now, by *hz* extensions I mean margin kerning and font expansion in pdfTEX.

2 How to start using *hz* extensions

Similarly as in the case with TEX primitives, the *hz* extensions as provided by pdfTEX are not easy to use. They require the understanding how certain concepts in TEX work at the very low-level, as well as the ability to set up some complicated font-related stuff. Thus the best way to start using *hz* extensions is via some available interface. An article about how to start using *hz* extensions in practice will be published in TUGboat soon so I don't go into the details here. In short, given that we have pdfTEX version at least 1.20a (1.21a is recommended at the time of writing this paper) and the LATEX package microtype installed, it's enough to say

```
\usepackage{microtype}
```

to activate both margin kerning and font expansion. A recommended next step is to read the microtype manual to learn more about the options the package offers, as well as advice for new users.

3 Practical experiences

hz extensions have been used rarely in practice, due to the lack of an easy interface and the necessity of complex font setup as mentioned above. Also, those features are not completely mature yet. For those reasons, a collection of practical issues would be helpful for those interested in using *hz* extensions. Some of the issues described here may sound very technical, or very weird to those who have not tried to use *hz* extensions in pdfTEX yet. If it is the case, simply skip the details you don't understand.

So far, the total number of people who have contacted me to discuss some issue concerning *hz* extensions is about ten. Some of them use *hz* for some occasional projects, some use *hz* for their regular work. Of course it doesn't have to mean that only ten people have been using *hz* extensions; it only means that at least ten people tried and had some trouble.

Recently, ConTEXt and LATEX (the pdfcprot and microtype package) make *hz* extensions easier to use and become more popular, especially margin kerning. The introduction of auto expansion is also a big step toward easy use of font expansion.

3.1 Using the auto expansion feature

To deploy font expansion, the expanded TFM fonts must be prepared ahead, using some utility as `fontinst`, afm2tfm or METAFONT. This is the most annoying part, even for experienced users. And usually this annoyance is doubled by the fact that most TFM's must be used with VF, hence the expanded VF's must be created as well.

[1]The *hz* program is in fact a set of modules implementing certain micro-typographic improvements including margin kerning and font expansion. *hz* extensions of pdfTEX are only a small subset of the modules in the *hz* program

From version 1.20a, pdfTEX supports a feature called auto expansion, which allows TFM and VF to be expanded automatically in memory at run time. That means expanded TFM and VF are no longer required, which is a great relief. In order to use font expansion now, it's enough to upgrade pdfTEX binaries, install the LATEX microtype package, and that's it. No need to deal with expanded TFM's and VF's, map files or whatever. This feature makes things really simple. The implementation however is not that simple and it took some time to evolve and mature.

There is one catch in using virtual fonts with auto expansion: in virtual fonts accented characters are often drawn as composition of two glyphs, a base letter and an accent. Using auto expansion will cause the accent in such composed glyph to be misplaced by a small amount (0.01–0.1pt).

Now one may wonder whether auto expansion should be used in case all the expanded TFM's and VF's already exist (most likely because the user had to create them manually when auto expansion was not available yet). The answer is if DVI output is not considered (see the next issue) and only Type 1 fonts are being used, auto expansion should always be used. This is also the typical case.

3.2 Using *hz* extensions in DVI mode

hz extensions are available in both PDF and DVI mode. Using *hz* in DVI mode is much similar to PDF, although it requires some extra setup for dvips to process the expanded fonts. If only margin kerning is used, then there is no difference whether it is used in DVI or PDF mode.

The question is why one would want to use *hz* in DVI mode? There are some known reasons:

1. the output is smaller: in PDF mode pdfTEX embeds many instances for a single expanded font, while dvips embeds only one;

2. the document requires PS processing, for example PStricks;

3. the user just doesn't need or like PDF, but wants *hz*.

The following (typical) example demonstrates how to make dvips process DVI files with font expansion enabled. It only makes sense to people who can already make some setup to use font expansion in PDF mode without auto expansion, which means that you must be able to create the expanded TFM's and VF's using `fontinst` or afm2tfm or some similar tool. The detailed instructions on how to do that is however out of scope of this article.

Assume that we have activated font expansion for font `cmr12` with stretch limit 20, shrink limit 20 and expansion step 5. To process the DVI file produced using this setup, we must update the map entry read by dvips. In my system, the entry for `cmr12` is

```
cmr12     CMR12  <cmr12.pfb
```

Now it must be replaced by

```
cmr12     CMR12  <cmr12.pfb
cmr12+5   CMR12  "1.005 ExtendFont"  <cmr12.pfb
cmr12+10  CMR12  "1.010 ExtendFont"  <cmr12.pfb
cmr12+15  CMR12  "1.015 ExtendFont"  <cmr12.pfb
```

```
cmr12+20  CMR12  "1.020 ExtendFont"  <cmr12.pfb
cmr12-5   CMR12  ".995 ExtendFont"   <cmr12.pfb
cmr12-10  CMR12  ".990 ExtendFont"   <cmr12.pfb
cmr12-15  CMR12  ".985 ExtendFont"   <cmr12.pfb
cmr12-20  CMR12  ".980 ExtendFont"   <cmr12.pfb
```

Then dvips can process the DVI with expanded fonts just like any other DVI files.

The main disadvantage of DVI mode is, however, that the auto expansion feature of pdfTEX cannot be used. Or to be more precise, auto expansion can be activated in DVI mode and pdfTEX can create the DVI file with font expansion enabled. Such a DVI file however is pretty useless because DVI drivers cannot process that file, as they don't have access to the expanded TFM's and VF's (those exist in pdfTEX memory at run time only).

There have been requests to support auto expansion for use with dvips, by changing pdfTEX to write the expanded TFM's and VF's to disk as well as to update some map files. It is likely that this will never be implemented. The better way to do that is to change the script TEX (pdfTEX) calls to create missing TFM at run time (on web2c-based systems this is called `mktextfm`) to create all the required stuff on demand.

3.3 Margin kerning and non-character material

Margin kerning only works with characters. Sometime we need to protrude something that is not a character, for example some superscript (index), because such a thing would look bad when ending up at the margin without protrusion. The typical example is the index of footnote, which is typeset into an hbox.

There are two solutions to the above problem:

1. Make margin kerning work even with characters inside boxes. A patch has been made to check whether the ending element of a line is a box, and if so check the last element of that box, and so on, to ensure that the last character inside boxes will get protruded. This patch is still experimental at the time writing this article. Also beware that it is not enough just to have this patch and the right package loaded, because the default settings for margin kerning as provided by macro packages are suitable for normal text. Superscript text often has much smaller size than the normal text, so to get the right result the protrusion factors must be increased. Then it might cause conflicts in other places where the same font is used for normal text (for example in footnote text).

2. Append a "virtual character" immediately after the material we want to protrude into the right margin (or prepend in case of the left margin). Such a virtual character is not visible, has no dimension, but has non-zero protrusion factor so when ending up at the margin it will get protruded. Creating such a virtual character is quite easy using `fontinst`. There is also a script available to generate a virtual font with all blank characters for this purpose[2]. This approach has been used several times in practice and is reliable. The drawback is that it requires some extra work in creating those virtual characters and inserting them into the right places.

[2]The script was made by Hartmut Henkel, however having read the draft version of this paper he suggested to distribute the font itself; so the font will be available at `pdftex.sarovar.org` soon.

3.4 Margin kerning does not work in some cases

Sometimes it happens that margin kerning doesn't work as expected, some characters are not protruded. There are usually two reasons:

1. Margin kerning is blocked by some invisible material around the relevant character; usually there is a workaround using macros.
2. It is a bug in pdfTEX. If your pdfTEX is older than 1.20b then upgrading pdfTEX is the first thing to consider when encountering some problem with *hz* extensions. Margin kerning has been improved a lot from version 1.20b.

3.5 Using margin kerning and font expansion at the same time

Although this seems something evident, sometimes it doesn't work. The reason is most likely that you are using pdfTEX version 1.20a, which has this bug.

3.6 Reasonable settings for font expansion

Font expansion must be used with care. While it gives more room for line-breaking, it can also destroy the whole text if the effect of font expansion becomes visible. Then the question is when font expansion becomes visible? This question has no definitive answer, as to trained eyes font expansion can be easily detected and hence annoying, while to others it has no effect. For most people the safe limit is 2 % expansion, i. e. the stretch and shrink limit when expanding a font should not be more than 20 (see pdfTEX manual for explanation of stretch and shrink limit).

The expansion step is usually set to 5. A too small value leads to large output size (more fonts embedded), while a too big value can lead to some surprises like unexpected overfull or underfull boxes. This is not a bug but a limitation in the way pdfTEX implements font expansion.

4 Lessons learned

During the development of *hz* extensions, many decisions were made and not all of them were good. pdfTEX started as an experimental project. Most decisions in the beginning were made rather for the purpose to examine the effect of *hz* extensions than for practical purpose. Then *hz* extensions started to be used in practice and certain things had to be changed to make life easier. Here are some lessons I learned concerning *hz* extensions:

1. The way margin kerning can be used without changing existing fonts seems to be the right decision. Settings for margin kerning should be part of fonts just like kerning between pairs of characters, from the viewpoint of clean design. Doing so however would lead to two problems:

   (a) to use margin kerning, we need some kind of extended TFM;

   (b) the settings cannot be changed easily.

   These problems would make practical use of margin kerning impossible. Clean design is important but sometime backward compatibility and flexibility play a more significant role.

2. The concept of font expansion in pdfTEX was quite general and flexible, which was good for experiments as we needed to study the effect of font expansion in as many cases as possible. From the experimental viewpoint the way font expansion was implemented is not that bad, but from practical viewpoint it is too cumbersome and hard to deploy. Hence for practical use, a less flexible mechanism which is easier to use but offers 80% of what font expansion can give is more needed. That's the reason why auto expansion has been introduced.

   So was it a bad decision that pdfTEX needed expanded TFM for font expansion? For experimental purpose it was a good decision, however it would be a mistake not to change it after the experiments have been done.

3. The user interface is as important as the implementation to end users. Without LATEX or ConTEXt support, *hz* extensions are pretty useless to most users.

4. Feedback and help from the pdfTEX user community is vital for *hz* extensions to evolve and mature.

5 Recent changes

pdfTEX version 0.14h is the last version of pdfTEX released by me during my stay in Czech republic. Then I came back to Vietnam and didn't have time to work on pdfTEX for about two years. During that time pdfTEX was maintained by the pdfTEX team (lead by Hans Hagen and Martin Schröder). Since March 2004 I came back to work on pdfTEX development, and version 1.20a is the first release I participated in after the long break.

Here I would like to give a brief summary of some noticeable changes since version 1.20a, as I think that they might be of interest to pdfTEX users or simply to those who don't use pdfTEX but want to keep an eye on what is happening with pdfTEX. Apart from those, there have been many small bug fixes and improvements but they are too technical to mention here.

1. *hz* extensions have been significantly improved; some serious bugs have been fixed and margin kerning has been extended to handle some special cases.

2. Auto font expansion has been introduced: expanded TFM's are no longer required to use font expansion.

3. The font expansion mechanism has been simplified: the font expand factor (the last argument of `\pdffontexpand`) is no longer supported. This parameter was used to simulate font expansion using letter spacing. Experiments have shown that this technique is not the way to go: it didn't improve much the result while it caused many troubles.

4. Support for the configuration file `pdftex.cfg` is gone; all parameters are set via primitives. Their values can be dumped to the format file. The only exception is `\pdfmapfile`: its value cannot be dumped to the format file, so when pdfTEX starts the value of `\pdfmapfile` is always set to the default value `"pdftex.map"`.

5. pdfTEX uses the GNU libAVL library to speed up certain searchings.

6. Support for TrueType fonts has been improved, allowing refering to glyphs inside a TrueType font by their unicode index. `ttf2afm` also has been heavily revised.

7. There is a program called pdfxTEX, which is a variant of pdfeTEX that contains experimental features. Those features may be moved to pdfTEX when they seem to be useful and stable. At the moment the following extensions are avaiable:

   - `\pdflastximagecolordepth` returns the last color depth of a bitmapped image;
   - `\pdfximage` supports a keyword `colorspace` following an object number representing a PDF ColorSpace object;
   - `\pdfstrcmp` compares two strings;
   - `\pdfescapestring` and `\pdflastescapedstring` provide a means to escape strings;
   - `\pdffirstlineheight`, `\pdflastlinedepth`, `\pdfeachlineheight` and `\pdfeachlinedepth` allow fixing line dimensions during paragraph buiding;
   - various extensions from Taco Hoekwater:
     - support for dimension unit `px`;
     - `\tagcode` primitive allowing read and write access to a character's `char_tag` info.
     - `\quitvmode` primitive quits vertical mode;

6 Pending requests and future development

In this section I would like to mention shortly some issues that have been discussed and the future plan of pdfTEX.

PDF inclusion with annotations: when pdfTEX includes a PDF figure, all the annotations (the PDF term for hyperlinks and the like) from the figure are lost. There is a patch for pdfTEX version 1.10b by Andreas Matthias that copies annotations from included PDF figures. The patch was released during the period when I was not maintaining pdfTEX and didn't get attention from other pdfTEX maintainers either. I am aware of the patch but did not look into the code yet. If this becomes urgent or frequently asked then it should be re-considered.

Implement virtual fonts using Type 3 fonts: pdfTEX supports virtual fonts in the same way like other DVI drivers does, i. e. it interprets the DVI commands from virtual fonts to draw characters. It means that accented characters from virtual fonts are often unsearchable – in fact there are no such letters in the PDF output but sequences of PDF commands drawing the base letter and the accent. There have been requests to make letters from virtual fonts searchable. One possible solution is to implement virtual fonts as Type 3 fonts in PDF. This feature is something very handy to have, as it can allows other nice things as well. At the moment this feature is still being examined, and might be supported in the future if the effort required to implement it is not too much.

Support for subfont scheme for use with huge TrueType fonts: Subfont scheme is a trick to split huge TrueType fonts (usually used for Asian languages like Chinese, Japanese or Korean) into smaller pieces so that they can be used with 8-bit TEX. pdfTEX has some support for subfont scheme, but still very poor. At the moment this is being revised and should be improved in the near future.

Support for pdfsync: pdfsync is a package allowing synchronization between a PDF file created by pdfTEX and its LATEX source: the user clicks on some point in the PDF file and the editor "jumps" to the corresponding place in the source. The current implementation of pdfsync still has some unsolved issues, due to the lack of low-level support in pdfTEX. We had some discussions with the pdfsync author and I plan to provide some hooks to support pdfsync.

7 Acknowledgments

Too many people helped pdfTEX development in various ways, so it is impossible to thank all people here without missing someone. However, I would like to say thanks to a few people whose impact on pdfTEX is most vital in the last years:

1. Hans Hagen for his testing, discussions, feature requests, feedbacks and encouragement;
2. Hartmut Henkel for his important contributions on pdfTEX development and maintenance, especially in *hz* extensions;
3. Martin Schröder for his effort on keeping pdfTEX in sync with latest sources of libraries and other pieces of TEXLive and teTEX; he is also responsible for official pdfTEX releases;
4. and NTG and DANTE for financial support on this work.

References

[Thành 2001] Hàn Thế Thành, *Margin Kerning and Font Expansion with pdfTEX*, in: *TUGBoat*, vol. 22(2001), no. 3 – Proceedings of the 2001 Annual Meeting, pp. 146–148. (Online at `http://www.tug.org/TUGboat/Articles/tb22-3/tb72thanh.pdf`)

Newmath and Unicode

Johannes Küster*

Abstract

The "Newmath" project aims at defining and implementing new standard encodings for math fonts, and at the development of accompanying tools and packages. Switching math fonts should be made as easy as switching text fonts. The project stopped in 1998, as efforts were concentrated on the definition of Unicode codepoints for mathematics.

This article outlines my ideas for further development of Newmath. It deals mainly with the "encodings and fonts" part of the project. Originally the project aimed only at extending and reorganising the encodings of existing math fonts, but its objectives should be widened now to Unicode math – to make all those mathematical characters accessible and usable in TEX-based systems. The last section gives an outlook on Latin Modern Math fonts development.

Introduction

About 12 years ago, the Math Font Group[1] (MFG) started a project to define new standard encodings for TEX math fonts, together with the development of fonts implementing this new standard and of accompanying tools and packages. The new encodings should bring an extension to 256 codepoints per font. They should become the standard for TEX math fonts, ideally making it just as easy to switch between different math fonts as it has been achieved for text fonts. This whole project is called "Newmath", short for New Math Font Setup ("newmath.sty" is the name of the principal LATEX package implementing the new math font setup).

The development of Newmath stopped about 6 years ago, as it was decided (at the EuroTEX conference in St. Malo, I think) to concentrate efforts first on "Math into Unicode", i.e. to identify all mathematical symbols in (reasonable) use and to get these symbols encoded in the Unicode standard. This goal has been achieved for quite a while now, mainly with Unicode 3.2 in 2002, but work on the Newmath encodings has not been resumed since (mainly because the people originally involved quit the project meanwhile).

Is further development of Newmath still interesting at all, despite Unicode and OpenType fonts? I think it is, for the reasons discussed below. But its initial objectives should be widened: to make all Unicode math characters accessible in TEX in a standard way, but also to make math fonts easier to design or to adapt, and to make them more usable for other typesetting systems.

* info@typoma.com; http://www.typoma.com

1 The Math Font Group is a joint venture of the LATEX3 project and the TEX Users Group Technical Working Group on Extended Math Font Encoding. For more information see the Math Font Group's homepage [1].

Current State of Newmath

Partial implementations of the new Math Font Encodings, information about the development of Newmath, links to articles, conference presentations, mailing list archives etc. can be found at the MFG homepage [1]. The development of Newmath stopped in 1998 with version 0.59a. The implementation was mainly done by Matthias Clasen with Ulrik Vieth.

Currently six encodings are defined:

- *Math Core* (MC) contains Math Italic, Greek Upright and Italic, basic delimiters and other "alphabetic" characters (i.e. most of the characters which are really dependent on the font design and/or which are most likely to pre-exist in a text font)
- *Math Symbol Principal* (MSP)[2] contains a Calligraphic (or Formal Script) alphabet, the most important mathematical symbols, and basic accents
- *Math Symbol 1* (MS1 or MSA) contains a Blackboard Bold (or "Doublestroke") alphabet and additional mathematical symbols
- *Math Symbol 2* (MS2 or MSB) contains a Fraktur (or Blackletter) alphabet, some additional delimiters and accents, and an "Arrow Kit" (consisting of left and right arrow endings and repeatable middle parts (with negated and gapped versions), by which a great variety of different arrows at any desired length can be composed)
- *Math Extension Principal* (MXP)[2] contains text and display versions of big operators and integrals, wider version of basic accents (hat and tilde), the larger and extensible versions of the basic delimiters, larger root symbols, over- and underbrace parts, parts for extensible vertical arrows and bars
- *Math Extension 1* (MX1 or MXA) contains additional big operators and integrals, larger and extensible versions of delimiters, and wider accents (vector, bar, tie, etc.).

Each of the Symbol encodings contains a complete alphabet (A-Z, a-z, digits 0-9, dotless i and j) of a specific design for use in mathematical typesetting.

Compared with TEX's original "Math Extension" encoding, the new extension font encodings offer a much wider range of wide accents (9 sizes of each accent) and large delimiters (7 sizes instead of 4 for most delimiters, 14 for parentheses and non-extensible delimiters like angle brackets, plus extensible parts as necessary).

Unicode Math

Since version 3.2, Unicode assigns almost 2.000 codepoints for mathematical characters. Due to the way in which Unicode evolved, and as new versions should be backward compatible, these codepoints are scattered over many Unicode blocks (mainly over 15 blocks in fact, of which 7 blocks are devoted exclusively to math characters; in addition to these another 15 blocks each contain a few characters for occasional use in mathematics).

2 The original name of the Math Symbol and Math Extension "Principal" encodings was "Primary", which was subsequently changed to "Privilege". As I think neither really conveys the intended meaning, I changed the name to "Principal" here.

Additional information about Unicode math is given in the Unicode Technical Report #25 [2] and in “MathClass6.txt” [3], a file which classifies the Unicode math characters according to their usage and provides “a mapping to standard entity sets commonly used for SGML and MathML documents”. The classification is comparable to TeX’s mathematical symbol classes, with the additional classes “diacritic” (which is not handled as a class by TeX, but by `\mathaccent`) and “fence” (an unpaired delimiter or a delimiter-like separator; normally treated as `\mathrel` in TeX).

Glyph Variants in Unicode. Some mathematical symbols did not get a codepoint of their own, instead they can be accessed as a combination of two Unicode codes (examples are shown below). This is the case for the negated version of many relators, for variants of negated relators (with a vertical negation slash instead of a slanted one), and for some symbols which are considered mere stylistic or typographic variants of others.

∉	U+2209	∈⃒	U+2208, 20D2	not an element of
≧̸	U+2267, 0338	≧⃒	U+2267, 20D2	negated greater-than over equal to
≨	U+2268	≨︀	U+2268, FE00	less-than but not equal to

Examples of Unicode variants and combinations: with U+20D2 “vertical line overlay” and U+0338 “combining long solidus overlay” (with and without encoded negated version), and with U+FE00 “Variant Selector 1”

These variants are shown in three tables in [2] (Table 2.5 there shows those relators with encoded negated form for which a variant with vertical stroke overlay can be realized by composition of base character and overlay; Table 2.6 shows those relators for which the negated form can only be realized by composition (i.e. the negated form is not encoded itself); and Table 2.7 shows all the currently defined glyph variants, which can be realized as a combination with “Variant Selector 1” U+FE00).

Future Additions to Unicode Math. Of course Unicode math can be and will be extended in the future. The Unicode Pipeline Table [4] shows some mathematical characters which will be included in the forthcoming Unicode version 4.1.0, due in March 2005. Also newly discovered and newly invented symbols will be standardized in future version eventually, when they are used by a considerable number of people. These possible extension have to be taken into account when designing new math font encodings for TeX.

Reasons for Newmath

To see more clearly how Unicode math could be made usable for TeX, and why Newmath could still be very helpful, let’s see what Unicode does offer and what it does not.

Unicode offers a very large set of mathematical characters, with a standardized code referring to each character. But the backward-compatibility leads to a quite unordered way in which characters are presented within Unicode. This is no problem for a computer program (e.g. for any automatic conversion program, workflow processes and the like), but it makes it difficult for users to search for a specific character, or for font designers to get an overview over all those math characters.

Unicode does not offer any sorting or ranking of mathematical symbols, nor much information about the meaning or usage of most characters. Also Unicode encodes only base characters, thus leaving all typographic variants aside which are needed in proper mathematical composition. Now theoretically all those glyphs could be defined in one large OpenType font, by assigning glyphs in the Private Use Areas (PUA) of Unicode, and/or by defining those glyphs as alternate forms of their base glyph via OpenType features. Unfortunately, it is not very likely that a standard way of PUA usage will evolve for math fonts. Also, the currently defined OpenType features are hardly suitable or sufficient for math fonts.

So I see many reasons why Newmath is still interesting and could be useful, despite Unicode and OpenType, and despite any successor of TeX which will be Unicode and OpenType capable:

– Newmath offers a standard interface for TEX (LATEX, ConTEXt).
  Currently almost each set of math fonts comes with its own encodings, which makes font switching very cumbersome.

– Newmath will offer all the typographical variants needed.
  This comprises most of the characters in extension fonts: larger and extensible delimiters, arrows and root symbols; text and display versions of big operators and integrals; wide accents. (This could be done in an OpenType font as well, of course.)

– Newmath will order, sort and rank mathematical characters.
  This will give a much better overview than it is possible in Unicode, making it comparatively easy to find a specific character, to judge its importance, etc.

– Newmath could serve as a guideline to font designers.
  Within Unicode, it is very hard for a font designer to identify the characters needed for mathematics, and to seperate indispensable math characters from less important ones. In fact, most font designers will be abhorred by the prospect of designing 2000 additional characters, of which most will be only seldom used.

  Here Newmath could offer a clearly arranged and well-ordered set. For example is the math asterisk $*$ (U+204E LOW ASTERISK) easily overlooked, as most fonts already contain an asterisk * (U+002A ASTERISK), but normally this glyph is not suitable for math, which needs a larger version, six-pointed and vertically centered at the mathematical axis.

– Newmath could be used to classify math fonts.
  Currently it is not easy to judge the usability of a specific math font, and to gain a quick overview of its glyph complement set. This could be made much easier by classifying the font according to the set of its supported Newmath encodings. For example, it should be quite easy then to see that a font has all the Unicode glyphs for Mathematical Logic.

Further Development of Newmath

For further development of Newmath, I see the following areas: *Encodings; macros; fonts; packages and tools; integration and interaction with other packages; additions and enhancements to TEX's mathematical typesetting engine.*

I am mainly concerned with encodings, macros and fonts here, and my ideas for these areas are detailed in the sections below.

The development of "packages and tools" will, for a good part, go hand in hand with the development of macros (for LATEX, ConTEXt and plain TEX). By "integration and interaction with other packages" I mean that Newmath should work with other math packages (e.g `amsmath` or `nath` in LATEX), but also that Newmath could borrow and integrate from other packages (e.g. macros in widespread use could be standardized). For possible "additions and enhancements to TEX's mathematical typesetting engine", see Ulrik Vieth's article [5]. Here I'm only dealing with these aspects in the way they influence possible encodings.

Of course I won't and can't do all the necessary work alone, so anyone who wishes to help and to contribute is invited to join the project. Also all steps in the development will be discussed on the Math Font Group's "math-font-discuss" mailing list.

Development of the Encodings

General Considerations. We have to take TEX's restrictions into account: only 16 families of math fonts are allowed in one formula (practically, this means in one document in most cases). Therefore the additional encodings should be designed in a way that minimizes the loading of additional fonts. In a TFM file, only 15 different non-zero heights and 15 non-zero depths are allowed. While one could cope with this for symbol fonts in most cases, it is a really troublesome restriction when it comes to extension fonts. This leads to the strange vertical placement of most glyphs in extension fonts, which hinders their usability outside TEX. But even within TEX, it could become inpossible to cope with for some fonts which differ in design from some of Computer Modern's assumptions. These restrictions should be overcome by any successor of TEX, maybe best with a new "math font metrics" format, but for the time being, the encodings should deal with them as good as possible.

But the encodings and macro packages should not be tight to closely to TEX and the current situation; they have to be flexible enough to be extendable, to other typesetting traditions like those of traditional Russian mathematical typography, and to font set which bring their own extensions and special macros, like the MathTime Pro fonts.

The Existing Encodings. I consider *Math Core* as fairly stable. About 5 characters could be moved to another encoding maybe. This would allow to include a few Roman characters like e, i, and maybe D (MathCore already contains "d", and these single Roman letters are quite common in mathematical typography; inclusion in MathCore would allow kerning with Math Italic letters); but this may sacrifice compatibility with the old math font setup.

Both *Math Symbol Principal* and *Math Symbol One* are stable as well, maybe with one or two questionable characters in each one, and with about 40 yet unassigned codepoints in MS 1. The additional Unicode symbols supply some obvious candidates for inclusion here.

For *Math Symbol Two*, I think that the Fraktur digits are a misunderstanding, these should be replaced by old-style (tabular) digits (the "Fraktur digits" currently included here stem from the Euler fonts, but apparently these are intended as Euler Roman old-style digits; while proper Fraktur digits – clearly visually seperate from Roman digits – just do not exist). The Arrow Kit could be moved to an encoding of its own, as many more arrow pieces could be added then.

The case is different for the *extension fonts:* maybe the whole encodings should be overthrown, maybe we should sacrifice compatibility with older documents here in favour of a clearer layout. By putting e.g. root, accents, and over- and underbrace into one encoding, and putting delimiters and big operators into a second one, most glyphs could be brought to their natural position, which would greatly improve on the design and general usability of such fonts.

New Additional Encodings. For the remaining Unicode math characters (i.e. for the characters not yet encoded in Newmath), we have to design new, additional encodings. First, let's see how many additional characters there are, and how many additional forms (like larger delimiters) we need. The following table gives a rough number for the additional characters in each class, with the number of additional codepoints needed in TEX:

Arrows:	250	+ arrow kit pieces:	100	
Binary Operators:	130			
Geometric Symbols:	100			
Miscellaneous Symbols:	30			
Ordinary Symbols:	90			
Punctuation:	15			
Relators:	230	+ negated variants:	100	
Z Notation:	10			
Accents and Overlays:	30	+ in extension fonts:		90
Big Operators:	0	+ in extension fonts:		40
Delimiters:	45	+ in extension fonts:		450
Integrals:	25	+ in extension fonts:		50
Total number of glyphs:	955	+ in symbol fonts:	200	
		+ in extension fonts:		630

This would mean 4 or 5 additional symbol font encodings (possibly including 1 or 2 arrow kit encodings), and 3 or 4 additional extension font encodings.

To minimize the loading of additional fonts, and to offer clearly arranged font layouts (both to users and to font designers), we should sort and group the Unicode characters, according to importance, meaning, and area of use (within mathematics). For example, all symbols specific to one field of mathematics should be kept together in one encoding. Examples are logic, geometry, or z-notation symbols.

Most documents will only need a limited set of mathematical symbols, and well-designed encodings should make it possible to keep within TEX's restriction to 16 math font families in most cases – without the need for mid-document changes of math encodings.

Unfortunately Unicode does not provide much information neither about the use of a specific character nor about its field of use, so many characters need some research before one could group them properly into an encoding.

Macros and Packages

Along with the new encodings, we do need standard macro names to access those glyphs. Again, any information about the meaning of a character is very helpful here, as then the macro could be name accordingly. For some characters, this is a rather straightforward task.

Ideally, these macro names should be the same in all TEX based systems (especially in plain TEX, LATEX and ConTEXt). Of course one has to develop a "Newmath" package (or file bundle) for each system, but these macro definitions will form the core of each such package and will essentially remain the same.

For LATEX, the current version of Newmath already supplies the necessary files, so these have to be extended accordingly. When Newmath development ceased in 1998, ConTEXt was not very widespread yet, but now Newmath should of course support it (and vice versa).

TEX's existing math macro names could be broadly categorized as

– descriptive (describing the shape, e.g. `\uparrow`)
– semantic (describing the meaning, e.g. `\sum`, `\times`)
– mixed (partly semantic, partly descriptive, e.g. `\otimes`).

Obviously Knuth employed the following scheme: any symbol with one fixed meaning gets a macro name according to its semantics (thus, `\sum` and not `\bigsigmaup` or something). Any symbol without fixed or with more-than-one meaning gets a macro name describing the shape. And the mixed names come in for symbols where the base symbol or a component of the symbol has a semantic name already (but where the meaning of the combined symbol is not clear or not fixed). Of course the new additional macro names should follow this scheme, using a semantic name whereever possible.

In addition, Newmath could be extended to gather macros in widespread use, which could be standardized by including them in Newmath packages. Examples of such macros are `\abs{...}` and `\norm{...}`. Supplying such a standardized set of (alternative) semantic macros could be very helpful to many users. In fact, by using well-chosen semantic macros, a TEXsource sometimes can become more readable than its pretty-printed output, and of course it greatly helps in conversion e.g. to Content MathML or OpenMath.

Latin Modern Math Fonts

Of course encodings, macro packages, and tools are only useful together with fonts. Freely available math fonts could be extended and reencoded, and commercial math fonts could be mapped to the new encodings via virtual fonts (but normally they will lack many of the additional glyphs). A good part of this work (on Computer Modern extension and on virtual fonts for other math fonts) has been done already in the last version of Newmath, of course based on the math fonts available at that time. For the future development, I think it won't be a very useful approach to extend the Metafont sources of Computer Modern, as most users would want PostScript Type 1 or OpenType fonts. Instead, I think of extending the Latin Modern fonts.

The Latin Modern math fonts will be a set of freely available math fonts, to be used with Latin Modern text fonts. These fonts will be developed by me (and with the help of anybody who volunteers to work on these fonts). However I did not start to work on these fonts yet, so I can only give an account of my ideas and intentions here.

I will do the development in MetaType1, so the resulting fonts will be in Type 1 format and could be wrapped as (CFF flavoured) OpenType, too. Eventually the MetaType1 sources will be released as well, just as it is done (or planned) with Latin Modern text fonts. In general, the design will follow Computer Modern math fonts, with 6 versions of each glyph: in 2 weights (regular and boldface), times 3 optical sizes (5 / 7 / 10pt, or rather "Tiny", "Caption", "Regular", as the fonts will be freely scalable of course). But the number of glyphs will be considerably extended, to comprise the complete set of Unicode math characters alongside all characters defined in Newmath.

Neither the design nor the metrics of the fonts will be completely compatible with Computer Modern: the design should be more "of a piece" than with Computer Modern and its various extensions (just one example of such a mis-match: the 3 Hebrew letters Beth, Gimel, Daleth from the AMS fonts do not match the design of CM's Aleph, they rather match the Euler fonts' Aleph); and metrics will be changed as needed (e.g. many new kerning pairs will be possibly due to the extended encodings).

By default, these fonts will be encoded in the Newmath standard, thus offering a freely available implementation of the standard. But by the MetaType1 approach the fonts will be independent of any de-facto encoding – thus one could rather easy adapt them to further Newmath development, to different encodings, or even to the requirements of other typesetting applications.

References

[1] Math font group project homepage. `http://www.tug.org/twg/mfg/`

[2] Barbara Beeton, Asmus Freytag, and Murray Sargent III. *Unicode technical report #25: Unicode support for mathematics.* `http://www.unicode.org/reports/tr25/`

[3] *MathClass-6.txt – Classification of math characters by usage.* `http://www.unicode.org/reports/tr25/MathClass-6.txt`

[4] *Unicode proposed new characters: The pipeline table.* `http://www.unicode.org/alloc/Pipeline.html`

[5] Ulrik Vieth. Math typesetting in TEX: The good, the bad, the ugly. In: *EuroTEX 2001* (proceedings of the 12th European TEX conference, Kerkrade, the Netherlands), pages 207–216, 2001.

Latin Modern fonts: how less means more

Bogusław Jackowski and Janusz M. Nowacki

Well, less is more . . .
Robert Browning, "Andrea del Sarto," 1855.

1 Introduction

The Latin Modern project was launched during the 13th European and 10th Polish TEX Conference, May 2002, Bachotek, Poland. The aim was to prepare a family of outline fonts, compatible with Computer Modern fonts ([8]), but, unlike CMs, equipped with a rich collection of diacritical characters.

At that time, two solutions to the problem existed: (1) Lars Engebretsen's AE (Almost EC, [3]) family of virtual fonts, based on Computer Modern fonts in PostScript Type 1 format released by AMS; (2) Vladimir Volovich's collection of PostScript Type 1 fonts, CM-Super ([15]). Engebretsen's approach has an important drawback — virtual fonts can be used only with TEX. From this point of view, Volovich's CM-Super fonts would be a better choice. The fonts were produced by Péter Szabó's TEXtrace ([12]) which, in turn, is based on Martin Weber's Autotrace ([16]). Volovich's achievement is really impressive, nevertheless we would cast doubt upon the quality of the outlines of glyphs. This is perhaps the intrinsic drawback of the autotracing approach. Moreover, there is a problem with the size of the CM-Super package. It contains more than four hundred fonts; the size of the PFB files is almost 60 MB. Finally, it is not easy to to repeat the process of the font generation if changes are needed, as manual tuning was involved. (A comprehensive discussion of alternative approaches can be found in [5], [6], and particularly [11].)

Finding the situation unsatisfactory, some representatives of European TEX Users groups decided to prepare yet another family of fonts, Latin Modern, being in a way a continuation of Engebretsen's approach, but going further: the aim was to comprise all existing Latin-based alphabets, not necessarily European. We were invited to lead the project, which we gladly accepted.

2 The initial stage of the LM project

Like Engebretsen, we decided to make use of the Computer Modern fonts in the PostScript Type 1 format, released by AMS. But being bent on working with human readable sources, we decided to employ our (anyway favourite) METAPOST-based program, METATYPE1 ([6], [7]). One of the modules of the METATYPE1 package is a converter from PostScript fonts to METAPOST sources. So, the first step was the conversion of PostScript Type 1 fonts to METAPOST sources that were to be manually adjusted.

The decision was not obvious at all. The main disadvantage of such an approach is the "freezing" of parameterization. As an alternative, we considered a conversion of METAFONT CM sources into METATYPE1-conforming ones. It turned out, however, that this method, although practicable, would be time-consuming. Taking into account that our main goal was the extension of the *standard TEX fonts* with diacritical characters, we abandoned eventually the idea of working with METAFONT sources, although access to the CM parameterization is provided (see section 4.1).

3 Interim stages of the LM project

The LM family of fonts has been developing evolutionary. Our main concern, as already mentioned, was the enhancement of the character set, but, as a result of pressure from users, the number of fonts also increased.

3.1 Serious matters and trifles

The number of glyphs per font grew from less than two hundred to more than six hundred. Also, the number of fonts grew. We started with 50 fonts (following Engebretsen); currently, the LM family contains 57 fonts. Among them are fonts that do not belong to the "Knuthian canon," for example, the bold companion for `cmssq8` and `cmssqi8` (prepared by Pierre A. MacKay) — see [5], p. 67, for the complete list of LM fonts. The increase of the number of glyphs resulted of course in the rapid growth of the number of kern pairs.

The augmentation of the LM family of fonts was certainly the most important part of the whole enterprise. We wrote several tools (mostly `awk` scripts) that helped to control the herd of glyphs and interdependences between them — it is unimaginably difficult to fiddle with dozens of thousands of glyphs and hundreds of thousands of kern pairs by hand.

As we pointed out in [5], the lion share of our time was spent on the struggle against tiny details and exceptions. We were not expected to come up with brand-new concepts. On the contrary, we had to comply with the established practice. This, as it turned out, necessitated looking closely into lots of various aspects.

One can call most of these problems trifles, but their amount, relating of course to the size of the project, created a real problem. Such "trifles" had to be carefully analysed and even if the result of the analysis was simply "let's do nothing," it took

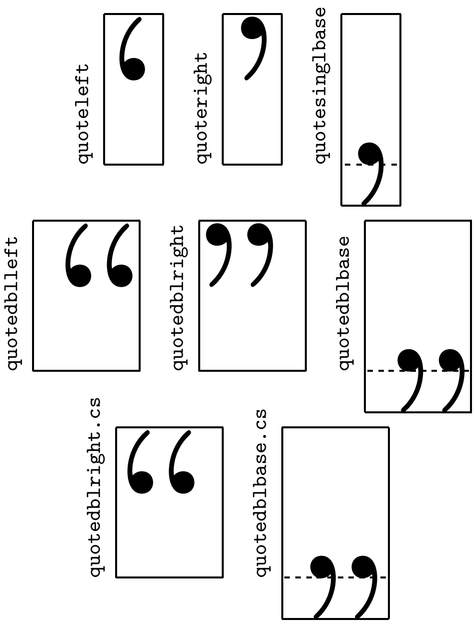


Figure 1: A seemingly innocuous asymmetry of double quotes turned out to be fairly bothersome; PostScript names of the glyphs have been used for the description.

time. Listing all details would be impracticable, but we cannot resist to mention just a few in order to demonstrate how seemingly less important things may cause more trouble.

3.2 Detail 1: asymmetry of double quotes

One of such problems turned out to be, somewhat unexpectedly, the asymmetry of double quotes (see figure 1). Observe that single quotes are positioned symmetrically, while double ones are not. This is the CM fonts heritage: the glyphs *quoteleft* (reverse apostrophe), *quoteright* (apostrophe), *quotedblleft* (opening quotes), and *quotedblright* (closing quotes) were designed by Donald E. Knuth ([8], p. 140 – 141 and p. 280 – 281), who decided to introduce asymmetry. But the glyph looking like an English opening quote is used in some languages, for example, Czech and German, as a closing one. Therefore, Czech TEX users introduced a special glyph with differently asymmetric sidebearings (*quotedblright.cs* in figure 1) in their variant of CM fonts. In consequence, the character *quotedblbase* (used as an opening quote, for example, in Czech, German, and Polish) also inherited the asymmetry[1].

[1] For historical reasons, the glyphs *quotedblbase* and *quotedblbase.cs* slightly differ; the latter is placed asymmetrically also in typewriter fonts.

The proliferation of glyphs caused by the asymmetry of *quotedblleft* and *quotedblright* is of course a disadvantage because fonts needlessly swell. We decided, however, to inflate them even more: symmetric quotes were provided as an alternative. We believe that only the latter ones should be used, but because of the remnants of history, the problem cannot be resolved once forever — asymmetric quotes should be retained.

3.3 Detail 2: non-uniform width of accents

Typically, accents should have the same width. This is, however, not the case with CM fonts: *cedilla*, *dotaccent* and *ring* have widths different from the remaining accents, that is, *acute*, *breve*, *caron*, *circumflex*, *dieresis* (umlaut), *grave*, *Hungarian umlaut*, *macron*, and *tilde*, all of which have the same width of $1/2$ em. We cannot say why *cedilla* and *dotaccent* are an exception. The idea behind the extraordinary width of *ring* can be easily understood if one inspects the code of the plain TEX ([9]) macro `\AA` which typesets the symbol Å (*Aring*). The glyph *ring* is designed to align with the top of the letter *A*:

```
\def\AA{\leavevmode\setbox0\hbox{!}%
  \dimen@\ht0\advance\dimen@-1ex%
  \rlap{\raise.67\dimen@\hbox{\char'27}}A}
```

It is tempting to have a unique width for all accents. But this would mean upward incompatibility, as the `\AA` macro would cease to work. On the other hand, it is not urgently needed, as *Aring* obviously belongs to the repertoire of LM glyphs. Perhaps the cure would be to introduce alternative accents and use the odd-sized ones only with TEX, and only when compatibility is needed, for example, if the LM fonts are to be used as a replacement for CMs (see section 4.2). But, as we complained already in section 3.2, a supererogatory increase of the number of glyphs would be an obvious disadvantage.

The plain TEX macro `\AA` is not the only one that heavily exploits the metric properties of CMs. The macros `\l` and `\L` (which define glyphs *lslash* and *Lslash*, respectively) also are defined in a CM-dependent manner. Both macros rely on the assumption that there is a special glyph in slot 32 (*suppress*; of course, the width of this "accent" glyph is different from a typical one) and that there are specific, unusually large kerns between this glyph and the letters *l* and *L*:

```
\def\l{\char32l}
\def\L{\leavevmode\setbox0\hbox{L}%
  \hbox to\wd0{\hss\char32L}}
```

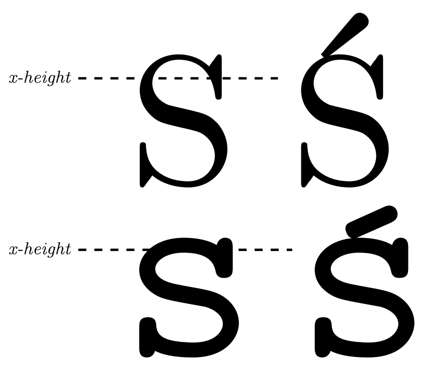


Figure 2: The lowercase letters in CM caps and small caps fonts, that is, `cmcsc10` (top) and `cmtcsc10` (bottom), are higher than the nominal *x-height*; this may befool some typesetting programs which may assume that lowercase letters should be accented without an additional vertical shift of the accent — the potential disastrous results are shown to the right; TEX rises an accent by the difference between the *ex* unit and the actual height of an accentee.

For example, the respective kern amounts in `cmr10` are −2.78 pt and −3.19 pt, while other kerns are generally in the order of a fraction of a point.

There are two intrinsic problems with *suppress*: (1) it does appear in worldwide standards, such as *Unicode* ([14]) or *Adobe Glyph List* ([2]), although *lslash* and *Lslash* appear there (the *Unicode* standard uses the name *stroke* instead of *slash*); (2) according to ASCII, slot 32 should be occupied by the *space* glyph. The result is easy to predict: most non-TEX fonts will not work properly with plain TEX and most non-TEX software will not work properly with standard TEX fonts...

3.4 Detail 3: bogus x-height in small caps

One might expect that *x-height* (that is, the *ex* unit in TEX) is approximately the height of the lowercase letter *x*. True, but it depends on the accuracy of approximation. The difference between the *x-height* and the height of the lowercase *x* is queerly large in CM caps and small caps (see figure 2). For example, in `cmcsc10` it reaches 0.83 pt. The answer to this riddle is simple: all roman fonts of the CM family have *exactly the same* value of *x-height*; in other words, `cmcsc10`'s *x-height* is the height of the `cmr10`'s lowercase *x*. This discrepancy is harmless, if not advantageous, for TEX, but if the fonts are to be used outside the TEX world, then one may expect weird results if a given system is capable of making composed characters. Nevertheless, we adopted the CM convention for the LM fonts in the hope that if a font is used in a different environment, all necessary characters that could potentially be composed will be already in it.

4 The present stage of the LM project

The project seems to be approaching a development plateau: in comparison with the state of the art reported in [5], the number of fonts was not changed, although the repertoire of characters has been augmented by approximately one hundred glyphs per font — now each font contains circa 650 glyphs. In particular, diacritical characters for Vietnamese and Navajo alphabets have been added — many thanks to Hàn Thế Thành, Karl Berry and Hans Hagen for their warm-hearted help.

As was mentioned, the LM family of fonts developed evolutionary. Everybody knows that evolution is capable to bring forth really bizzare creatures. So were the METATYPE1 sources of the LM fonts after two years of evolution. Two years more — and we would be lost in them. Hence the decision to repeat the initial step: the METATYPE1 sources were once more generated from the now current LM fonts in PostScript Type 1 format. Of course, manual tuning was again necessary, as the structure of automatically generated sources is not always fit for a particular purpose. The newly generated sources turned out to be satisfactorily legible, so we decided to release them publicly. Thus, one of the project's main goals was reached.

4.1 Structure of the LM sources — an overview

The LM family of fonts consists of a few general purpose files and files containing specific data for every font (see the listing of a sample driver file below). The data for each font is split into five files that contain:

- metric data,
- PostScript-oriented data,
- encoding data,
- the definition of shapes of basic glyphs,
- the information about ligatures and kerns.

All these files are governed by a single driver that inputs them — see lines 5, 6, 7, 10, and 13 in the listing below:

```
1 % A driver file for lmb10 Latin Modern font
2 input fontbase;
3 vardef cm_pal = "cmb10" enddef;
4 input comm_mac;   % common defs, CM params
5 input lmb10.mpm;  % metric
6 input lmb10.mph;  % PS-oriented header
7 input lmb10.mpe;  % encoding
8 input comm_mph;   % common header
9 beginfont
```

```
10 input lmb10.mpg;  % ‘‘frozen’’ glyphs
11 input comm_mpg;   % common glyphs (diacritics)
12 if known generating:
13  input lmb10.mpl; % ligatures and kerns
14 fi
15 endfont
```

There is no parameterization in these files. All entities are defined using bare numbers. The files are assumed to be “frozen” and are not expected to be altered in the future, unless new basic characters are added or severe bugs are spotted. The exception is, of course, the encoding file (line 7) that can be modified as need be.

Each LM font is associated with its CM pal (line 3). The respective CM driver file is being read and its parameters are stored for further use; they are exploited, for example, in the file `comm_mpg.mp` (line 11) by the programs defining the characters depicted in figure 3.

The `comm_mpg.mp` file is actually a “pivot” of the LM fonts. Its main purpose is to define accented glyphs, that is, diacritical characters that can be defined as composites. But not only. The file begins with three peculiar inputs:

```
input gly_euro.mp;
input gly_guil.mp;
input gly_vspa.mp;
```

The files being input are exceptional as they do not define accented characters. They contain a parametric METAFONT-based code for the following glyphs: *euro* (`gly_euro.mp`), *guillemotleft*, *guillemotright*, *guilsinglleft*, *guilsinglright* (`gly_guil.mp`), and *visible space* (`gly_vspa.mp`). The selection of glyphs is more or less arbitrary. The glyphs could be “frozen” as well; however, we decided to leave them in order to demonstrate what the METAFONT code would look like after a manual conversion to the METATYPE1 jargon.

Next, the definitions of letters *i* and *j* come. If one is surprised, one shouldn’t. After all, the letters *i* and *j* are simply *dotlessi* and *dotlessj* accented with *dotaccent*.

Then, the main part of the `comm_mpg.mp` file ensues. It reads as follows:

```
%% \vb\- Aacute:\- \PICT{Aacute}\-
acc_glyph(_A)(_Acute)(_Aacute);

%% \vb\- aacute:\- \PICT{aacute}\-
acc_glyph(_a)(_acute)(_aacute);

%% \vb\- Abreve:\- \PICT{Abreve}\-
acc_glyph(_A)(_Breve)(_Abreve);

%% \vb\- abreve:\- \PICT{abreve}\-
acc_glyph(_a)(_breve)(_abreve);
```

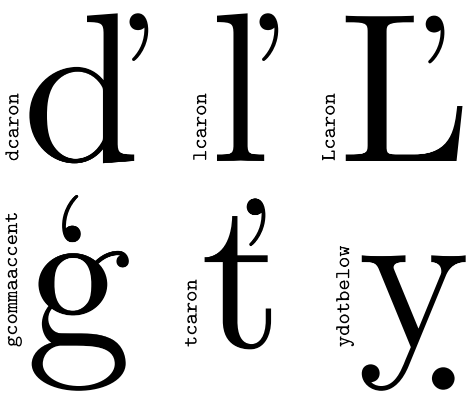


Figure 3: A group of diacritical characters in the LM fonts for which accents were positioned by hand. Note that although the suffix *caron* misleadingly appears in the glyph names, it is not an element of the respective glyphs.

```
%% \vb\- Abreveacute:\- \PICT{Abreveacute}\-
acc_glyph(_A)(_breveacute)(_Abreveacute);
...
```

The details of the code are unimportant — the reader is expected to understand what is going on here without arcane knowledge of the METAFONT language. We only mention that the persistently appearing macro `acc_glyph` automatically generates an accented character and that lines beginning with a double percent are meant for the preparing of proofs of a font.

The code looks a bit boring. Indeed, the majority of diacritical characters are composed using the macro `acc_glyph` which roughly corresponds to the TEX `\accent` primitive. In particular, the Vietnamese diacritics are defined in this way (see [4] for the details concerning the Vietnamese alphabet).

Note that there are different accents for uppercase and lowercase letters, for example, *Acute* and *acute*. Note also that double accents in the LM fonts, such as *breveacute*, are not defined using the macro `acc_glyph`, that is, they are supposed to belong to the basic set of glyphs. They might have been defined as composed objects, but this would increase the complexity of the fonts (for example, *abreveacute* would depend on *a* and *breveacute* and the latter, in turn, would depend on *breve* and *acute*), which we wanted to avoid. Moreover, in some cases subtle adjustments were needed. Therefore, we decided to “freeze” the double accents, once they had been created.

There are, however, several glyphs that cannot be obtained in a simple manner (see figure 3).

In the LM fonts, the following glyphs are specially programmed: *dcaron*, *gcommaaccent*, *Lcaron*, *lcaron*, *tcaron*, and *ydotbelow*. A punctilious reader may wish to examine the source code for the details of the implementation.

The final part of `comm_mpg.mp` defines duplicated glyphs, that is, glyphs of the same shape, but different names. For example, we decided to keep the glyphs named *Tcedilla* and *tcedilla* for historical reasons, although their proper names are *Tcommaaccent* and *tcommaaccent* (see [5], p. 70 – 71). Such duplication increases of course the size of a font, but not excessively. As already mentioned in [5] (p. 71), the duplication of a character adds only 30 – 40 bytes to a font. This is done by a METATYPE1 module which compresses PostScript Type 1 fonts. The module defines multiple occurrences of the same PostScript code as subroutines. In particular, whole characters can be defined as subroutines. This means that only the code that invokes these subroutines is to be added. Thus, the duplication of glyphs is moderately harmful which does not mean that it is always reasonable. In future, some of the duplicated glyphs might be deleted.

4.2 Using LM fonts with other `tfm` files

Obviously, as shown in [5], full LM and CM font compatibility can not be expected, that is, LM metric files cannot be used instead of CM ones. Still, it is possible to use LM fonts as a replacement for a subset of CMs: one should use CM metric files, a few special encoding files and a special font map file for the `dvips` driver. A typical line (broken here into two lines for technical reasons) from the relevant font map file for CM fonts looks as follows:

```
cmb10 LMRomanDemi10-Regular
    "enccmrm ReEncodeFont" <cmrm.enc <lmb10.pfb
```

This line says that TEX should use the metric file `cmb10.tfm` for typesetting, while the `dvips` driver should embed files `lmb10.pfb` and `cmrm.enc` instead of respective files for the CM fonts. Because the dimensions of the glyphs occuring both in the LM and CM fonts are the same (within the accuracy of rounding errors) and glyph shapes are very similar to each other, a user should not notice any difference, unless there is a bug in the LM fonts.

We hope that this solution will prove sufficient in most of practical cases. Similar files are provided for the PL and CS fonts, that is, for the Polish and Czech variants of the CM fonts. At present, work is being done on the support for VNS, that is, the Vietnamese variant of the CM fonts.

4.3 LM fonts in the *OpenType* format

The PostScript Type 1 format is claimed to be obsolete since many years. Actually, all PostScript engines support Type 1 fonts and are expected to support them also in the future. Recently, however, the *OpenType* format becomes a worldwide-accepted standard (see, for example, [10]). We believe that the TEX world should acquiesce to this. Therefore, we also prepared the collection of the LM fonts in the *OpenType* format.

The current release of the *OpenType* LM fonts should be considered experimental, although we gathered some experience during the preparation of the *OpenType* fonts for the *Antykwa Toruńska* family. We employed the *Adobe Font Development Kit for OpenType* (free but not open; see [1]) for the conversion from the PostScript Type 1 to *OpenType* format. An alternative could be *FontForge*, a marvellous openware font program by George Williams (see [17]). Currently, AFDKO better suits our purpose, but as *FontForge* is being constantly developed we hope to switch to it before long.

One of the most important innovations introduced in the *OpenType* format are so called *features*. These are tags that provide additional information about how to use the glyphs in a font. So far, five features have been built into the *OpenType* LM fonts:

- `cpsp` (*Capital Spacing*).
- `dlig` (*Discretionary Ligatures*),
- `frac` (*Fractions*),
- `liga` (*Standard Ligatures*),
- `onum` (*Old Style Numerals*).

The availability of these features depends upon application support, for example, the *Adobe InDesign* program under the control of the *Microsoft Windows 98* operating system offers all of them (see figure 4), while *Microsoft Word 2002* in the same system ignores *OpenType* features.

Perhaps the toughest problem is the grouping of LM fonts into subfamilies. The idea of a series of point sizes, as implemented by Knuth in the CM fonts, seems to be athwart the nowadays praxis. Nevertheless, we followed the Knuthian tradition — see figure 4. Feel warned, however, that the adopted grouping may likely change after consultations with experienced *OpenType* users.

5 Conclusions

As one can infer from the title of the paper, our aim was to obtain a product handy in use at the price of abandoning features that — as far as we perceive it — are only moderately usable.

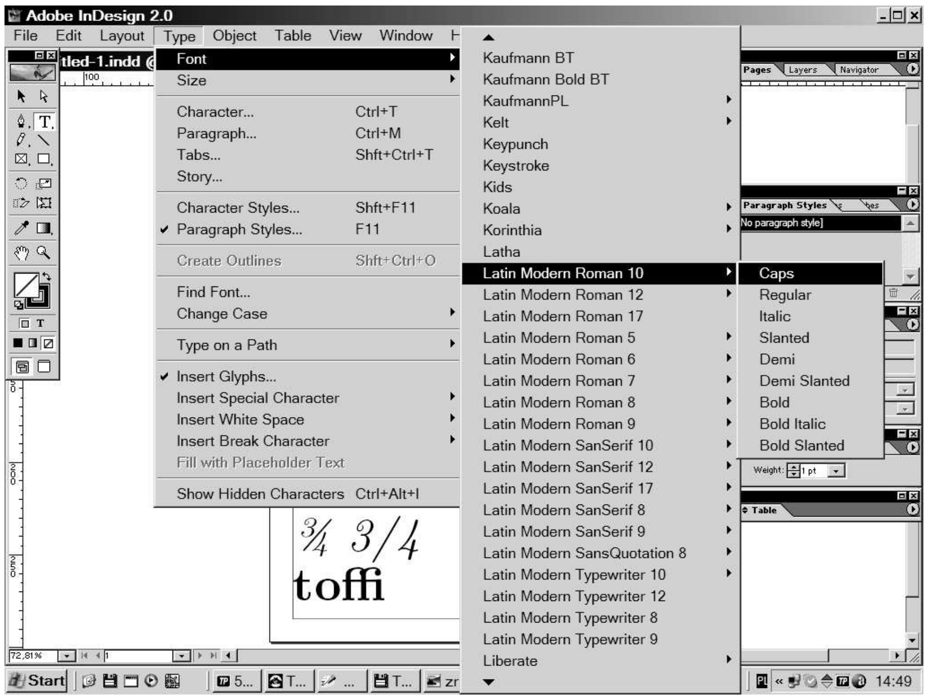


Figure 4: The *OpenType* LM family of fonts as seen by the *Adobe InDesign* program. Note that *Adobe Type Manager* used to accept subfamilies that contained at most four variants of a font, that is, normal, italic, bold, and bold italic. This is no longer the case with *OpenType* fonts — see the list displayed in the right part of the screen shot.

Just two examples:

- There are some fonts in the CM family that we never happened to use: `cmff10` and `cmfi10`. We decided not to include such fonts into the LM family.
- We did not follow the idea underlaying the EC fonts to provide a complete series of font sizes (our arguments are set forth in [5], p. 66), unlike Volovich with his CM-Super fonts.

It is for the users to judge whether the goal was achieved.

Actually, the LM family of fonts in many respects offers simultaneously less and more, not always unequivocally, for example:

- The number of fonts is less than in the CM family, but the repertoire of characters in each font is much larger.
- The LM parameterization is limited in comparison with the CM one, but we expect the potential modifications and augmentations of LMs to be easier, although the LM sources are much larger (6 MB after a compression) than the CM ones. Note that LM fonts take up much less space than the CM-Super ones (but reckoning with the uncompressed LM sources, the sizes become comparable).
- If the LM family would become a basic set of fonts for TEX (which we hope for), then the national variants of the CM fonts (PL, CS, VNS) could be dismissed, which would introduce more order into the TEX font distribution.
- The area of possible applications for the LM fonts is broader in comparison with the TEX fonts available so far, because of the furnishing of the LM distribution with the *OpenType* format.
- Although the LM glyph repertoire is already fairly rich, it can and should be extended further: the next step will be perhaps the addition of glyphs specific for African Latin-based alphabets (cf. [13]). It is not within the scope of the project, however, to include Cyrillic and Greek alphabets.

There is, however, at least one case where more means a not wanted more: the significantly large repertoire of glyphs per font means that the one-to-one correspondence between an LM font and its metric no longer exists and that a multitude of font metric files can be generated for a given font. This abundance is not necessarily what we want. But as long as TEX accepts 1-byte fonts only, the situation cannot be improved. This, however, is quite a different story.

The LM project is not finished yet. As all of us were taught by Donald E. Knuth, the debugging of software is a never-ending task and therefore software projects never end. But apart from fixing bugs, when the LM project reaches the stage of stability of metric data, we will consider the project essentially finished. Having legible sources, we are fairly optimistic — as far as we can assess, the LM project is rather more than less accomplished.

The current version, 0.98, of the LM fonts distribution is available either from CTAN or from `ftp://bop.eps.gda.pl/pub/lm`.

6 Acknowledgements

The Latin Modern project is supported by European TEX Users Groups, in particular by the German-speaking TEX Users Group DANTE e.V., the French-speaking TEX Users Group GUTenberg, the Polish TEX Users Group GUST, the Dutch-speaking TEX Users Group NTG, and, last but not least, by TUG — very many thanks. We would like to express our gratitude also to Harald Harders for preparing and maintaining the web page “Wishes for Latin Modern” (`http://www.harald-harders.de/latex/lmodern.html`) and to Jurek Ludwichowski for his willingly offered help, not only during the preparation of this paper.

References

[1] *Adobe Font Development Kit for OpenType*, `http://partners.adobe.com/public/developer/opentype/afdko/topic.html`

[2] *Adobe Glyph List, ver. 2.0, September 20, 2002*, `http://partners.adobe.com/public/developer/en/opentype/glyphlist.txt` and *Adobe Glyph List For New Fonts, ver. 1.1, April 17, 2003*, `http://partners.adobe.com/public/developer/en/opentype/aglfn13.txt`

[3] Lars Engebretsen, *AE fonts*, `http://ctan.org/tex-archive/fonts/ae/`

[4] Hàn Thế Thành, *Making Type 1 fonts for Vietnamese*, *TUGboat* 24(1), Proc. of the 24th Annual Meeting and Conference of the TEX Users Group, p. 69 – 84

[5] Bogusław Jackowski, Janusz M. Nowacki, *Enhancing Computer Modern with accents, accents, accents*, *TUGboat* 24(1), Proc. of the 24th Annual Meeting and Conference of the TEX Users Group, p. 64 – 74

[6] Bogusław Jackowski, Janusz M. Nowacki, *Programming PostScript Type 1 Fonts Using METATYPE1: Auditing, Enhancing, Creating*, Proc. of 14th EuroTEX, June 24th – 27th 2003, Brest, France, p. 151 – 157

[7] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *METATYPE1: A METAPOST-based Engine for Generating Type 1 Fonts*, Proc. of EuroTEX 2001, 27th – 27th September, 2001, Kerkrade, the Netherlands, p. 111 – 119; the current version of METATYPE1 is available from `ftp://bop.eps.gda.pl/pub/metatype1`; METATYPE1 for Linux prepared by Wlodek Bzyl can be downloaded from `ftp://ftp.ctan.org/tex-archive/systems/unix/mtype13/`

[8] Donald E. Knuth, *Computer Modern Typefaces*, Computers & Typesetting / E, Addison Wesley, 1986

[9] Donald E. Knuth, *plain.tex*, `http://www-cs-faculty.stanford.edu/~knuth/plain.tex.gz`

[10] *OpenType specification version 1.4*, `http://www.microsoft.com/typography/otspec/`

[11] Karel Píška, *Creating Type 1 fonts from METAFONT sources: Comparison of tools, techniques and results*, Preprints for the 25th Annual TUG Meeting, August 30th – September 3rd 2004, Xanthi, Greece, p. 54 – 64

[12] Péter Szabó, *TEXtrace*, `http://www.inf.bme.hu/~pts/textrace/`

[13] Conrad Taylor, *Typesetting African languages*, `http://www.ideography.co.uk/library/afrolingua.html`

[14] *The Unicode Standard 4.0. Final Unicode 4.0 names list*, `http://www.unicode.org/Public/UNIDATA/NamesList.txt`

[15] Vladimir Volovich, *CM-Super Font Package*, `ftp://ftp.vsu.ru/pub/tex/font-packs/cm-super/`

[16] Martin Weber, *Autotrace*, `http://autotrace.sourceforge.net/`

[17] George Williams, *FontForge — An outline font editor*, `http://fontforge.sourceforge.net/`

◇ Bogusław Jackowski
BOP s.c., Gdańsk, Poland
B_Jackowski@gust.org.pl

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

No abstract available

ProTEXt, a new TEX-Collection for Beginners

Thomas Feuerstack

February 27, 2005

Abstract

One of TEX's largest strengths is embedded in the high modularity of the program. Beside the processor itself, every TEXnican might use the Editor, Post-Processor-Program, etc. he prefers most. For beginners or only interested person this advantage can lead to difficulties, especially in times, where users have gotten accustomed at "complete environments".

ProTEXt shows a way, how even Beginners can easily setup a complete running system and therefore it eliminates one of the main obstacles in using TEX.

Bibliography Styles Easier with MlBibTEX

Jean-Michel HUFFLEN
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

We emphasise and discuss some methodology about writing bibliography styles using the nbst language, part of MlBibTEX. Most of the given tricks can also be applied to developing styles using XSLT, since nbst extends it closely. Last we show that the organisation of a bibliography style in several files allows modular decomposition.

Keywords: bibliographies, methodology, bibliography styles, multilingual features, BibTEX, MlBibTEX, bst, nbst, XML, XSLT.

Résumé

Nous dégageons et argumentons quelques méthodes d'écriture de styles bibliographiques au moyen du langage nbst de MlBibTEX. La plupart des conseils donnés peuvent également s'appliquer au développement de styles en XSLT, le langage nbst en étant assez proche. Enfin, nous montrons en quoi l'organisation des divers fichiers d'un style bibliographique permet une décomposition modulaire.

Mots-clés : bibliographies, méthodologie, styles bibliographiques, multilinguisme, BibTEX, MlBibTEX, bst, nbst, XML, XSLT.

Zusammenfassung

Es werden einige Methoden dargelegt und untergesucht, um bibliographische Styles in der Sprache nbst zu schreiben. Da nbst mit XSLT nah verwandt ist, kann diese Anleitung auch für die Programmierung der Styles in XSLT helfen. Am Ende wird an der Aufteilung der bibliographischen Styles in einzelne Dateien gezeigt, dass eine modulare Dekomposition möglich ist.

Stichwörter: Bibliographien, Methodik, bibliographischen Styles, mehrsprachigen Funktionen, BibTEX, MlBibTEX, bst, nbst, XML, XSLT.

Introduction

This article aims to give some methodology about the development of *bibliography styles*, that is, specifications that rule the layout of *references* put in the 'Bibliography' section of a document, these references being built from *entries* located in bibliography data bases.

When we started the development of our program MlBibTEX (for '**M**ulti**L**ingual BibTEX') [9], we were interested in going thoroughly into multilingual aspects for a bibliography processor belonging to the programs of TEX's family and especially, generating bibliographies as source files for the LATEX word processor [22], like BibTEX [26]. More precisely, we aimed to put into action an 'extended' BibTEX with multilingual features comparable with LATEX's. Another example of such an extension is given by the babelbib package and the bibliography styles in interface with it [7].

As we explained in [12], we think that such organisation — adopted for MlBibTEX's first version [9] — leads to complicated bibliography styles, since the language bst [25], used within BibTEX, is not modular: each style is a monolithic program put in

```
@INPROCEEDINGS{thys1997,
        AUTHOR = {first => Frank,
                  last => Thys},
        TITLE = {Auf der {Spur} des
                 {Vernichters}},
        BOOKTITLE = {Dinoland},
        EDITOR = {first => Wolfgang,
                  last => Holbein},
        PAGES = {353--437},
        PUBLISHER = {Bastei L\"{u}bbe},
        ADDRESS = {Bergisch Gladbach},
        YEAR = 1997,
        MONTH = aug,
        LANGUAGE = german}
```

Figure 1: Entry using MlBibTEX's syntax.

only a single file, so if we would like to add multilingual features, we have to extend each style separately. This point and others decided us to develop a new language, so-called nbst, for '**n**ew **b**ibliography **st**yles', close to XSLT[1], the language of transformations for XML[2] documents. We think that such a choice is good, since XML becomes a central formalism for document interchange. In particular, using nbst eases the production of bibliographies for XML documents: for instance, documents written using XSL-FO[3] [37], a language for describing high-quality print outputs, or DocBook [38], a system for writing structured documents.

We explain in [17] why MlBibTEX does not use XSLT itself, after converting bibliography (.bib) files into an XML-like format, as programs like BibteXML [6] or BIB2XML [27] do. However, if we agree to consider an XSLT-like language for bibliography styles, we have to rewrite most of the bibliography styles of BibTEX, if we want to provide some continuity with this program. There exists a way to import bst functions into an nbst program [11], nevertheless it is obvious that complete rewriting is prefereable, in order to take as much advantage as possible of this programming paradigm. We put some methodology into action to rewrite BibTEX's bibliography styles, we are giving these methods hereafter.

We begin with a small example, in order to illustrate the expressive power of nbst. Second we show how to design the layout of a reference. We consider a particular case: the `@INPROCEEDINGS` entry type of BibTEX — for an article in a conference proceedings or a story in an anthology — but our method is easily adaptable to any entry type. Then we implement our specification. Last, we show how to organise the different items of a bibliography and give some advice about the decomposition of an nbst program into several files. A succint comparison between bst and nbst statements is given as an annexe, followed by some complements about writing external functions using Scheme — the language used for developing MlBibTEX [15] — close to the expression language used as part of DSSSL[4] [18], the language of stylesheets of SGML[5] [8].

```
<inproceedings id="thys1997" language="german">
  <author>
    <name>
      <personname>
        <first>Frank</first><last>Thys</last>
      </personname>
    </name>
  </author>
  <title>
    Auf der <asitis>Spur</asitis> des
    <asitis>Vernichters</asitis>
  </title>
  <booktitle>Dinoland</booktitle>
  <editor>
    <name>
      <personname>
        <first>Wolfgang</first>
        <last>Holbein</last>
      </personname>
    </name>
  </editor>
  <publisher>Bastei Lübbe</publisher>
  <year>1997</year>
  <month><aug/></month>
  <address>Bergisch Gladbach</address>
  <pages>
    <firstpage>353</firstpage>
    <lastpage>457</lastpage>
  </pages>
</inproceedings>
```

Figure 2: The entry of Figure 1 as an XML tree.

What knowledge is required to read this article? A basic one about XML, XPath — the language used to address parts of an XML document — and XSLT is sufficient to just understand the examples given hereafter. Good introductions to them are [29, 30, 34], the 'official' references about XPath and XSLT, issued by the W3C[6], are [36, 35]. Concerning

[1]e**X**tensible **S**tylesheet **L**anguage **T**ransformations.
[2]e**X**tensible **M**arkup **L**anguage.
[3]e**X**tensible **S**tylesheet **L**anguage — **F**ormatting **O**bjects.
[4]**D**ocument **S**tyle **S**emantics and **S**pecification **L**anguage.
[5]**S**tandard **G**eneralized **M**arkup **L**anguage, the ancestor of XML. Now it has just historical interest.
[6]**W**orld **W**ide **W**eb **C**onsortium.

```
<!ELEMENT pages (onepage+ |
                 (firstpage,(ff | lastpage)) |
                 pages-verbatim)>
<!ELEMENT onepage         %INTEGER;>
<!ELEMENT firstpage       %INTEGER;>
<!ELEMENT lastpage        %INTEGER;>
<!ELEMENT ff              EMPTY>
<!ELEMENT pages-verbatim  (#PCDATA)>
<!--  Strictly speaking, '%INTEGER;' is a parameter
      entity (cf. [29, pp. 163–164]) standing for parsed
      character data ('#PCDATA'). But we use it for
      sake of readability, whenever the content of a
      text node is an integer, because DTDs'
      formalism does not know this type. 'ff' is for
      an unspecified number of following pages.
  -->
```

Figure 3: Excerpt from our DTD: specification of pages from a journal or book.

MlBibTEX more precisely, all its elements and functions used within path expressions are described in [13]. On another point, we think that developing new functions in Scheme by MlBibTEX's end-users is only needed for very specific applications, so referring to an introductory book such as [32] is sufficient to understand the given examples. MlBibTEX has been developed using the fifth revision of this language [19].

A small example

Let us consider the bibliographical entry given in Figure 1. Even if it roughly looks like a BibTEX entry, we can notice the use of syntactic features specific to MlBibTEX: a `LANGUAGE` field[7], some keywords for introducing the different parts of a person name: '`first`', '`last`'. All these syntactic features are described precisely in [13].

If this entry is cited throughout a document, the corresponding bibliographical reference, to be put at the 'References' section, looks like:

> [1] Frank Thys. Auf der Spur des Vernichters. In Wolfgang Holbein, editor, *Dinoland*, pp. 353–437, Bergisch Gladbach, August 1997. Bastei Lübbe.

We got this result by using 'old' BibTEX, operating on an 'old' bibliography (.bib) file. The bibliography style used above is plain.bst, that is, items are labelled by numbers, and first names are not abbreviated. This reference is supposed to be put at the end of a document written in English. If a German-speaking plain bibliography style — e.g., dtk.bst, used for the articles of the journal of the DANTE[8] group, *Die TEXnische Komödie* — is chosen, that results in:

> [1] Frank Thys: *Auf der Spur des Vernichters*; in *Dinoland* (Hg. Wolfgang Holbein); S. 353–437; Bergisch Gladbach; Aug. 1997; Bastei Lübbe.

so the stylistic differences between these two examples — for example, '.' after the author's name in English, ':' in German and French — shows that the layout of such references is language-dependent, in the sense that it is influenced by 'national' traditions.

When MlBibTEX parses the entry of Figure 1, the entry is processed as if it was the XML tree given in Figure 2; in fact, it results in the SXML[9] representation of such an XML tree. We can notice that this choice allows us to structure information given in some fields, for example, person names, in the `AUTHOR` and `EDITOR` fields, but also the first and last pages of a story belonging to an anthology, in the

```
FUNCTION {multi.page.check}
{ 't :=    % t is given the value of the PAGES field,
           % popped from the stack.
  #0 'multiresult :=    % I.e., multiresult ← false.
    { multiresult not   % While multiresult is
      t empty$ not      % false and t non-empty,
      and               % do
    }
    { t #1 #1 substring$        % compare t's first
      duplicate$ "-" =          % character with
      swap$ duplicate$ "," =    % '-', ',', '+';
      swap$ "+" =
      or or
        % if success, update multiresult;
        { #1 'multiresult := }
        % if not, update t by removing its head:
        { t #2 global.max$ substring$ 't := }
      if$
    }
  while$
  multiresult   % pushed result.
}
```

Figure 4: How BibTEX detects that several page numbers are given.

[7]Also used in conjonction with the mlbib package [23] or the natbib package [7], but in MlBibTEX, the corresponding values need not to be surrounded by braces or double-quote characters.

[8]*Deutschsprachige Anwendervereinigung TEX e.V.*

[9]Scheme implementation of XML, described in [20]. See [15] for more details about its use within MlBibTEX's implementation.

```
<nbst:template match="pages">
  <nbst:param name="beginning"/>
  <nbst:param name="ending"/>
  <nbst:value-of select="$beginning"/>
  <nbst:variable name="onepage-elements" select="onepage">
  <nbst:choose>
    <nbst:when test="$onepage-elements">
      <nbst:choose>
        <nbst:when test="count($one-page-elements) = 1"><nbst:text>\bblp</nbst:text></nbst:when>
        <nbst:otherwise><nbst:text>\bblpp</nbst:text></nbst:otherwise>
      </nbst:choose>
      <nbst:apply-templates select="$onepage-elements[1]"/>
    </nbst:when>
    <!--  Otherwise, firstpage element, followed by either the ff or a last page.      -->
    <nbst:otherwise><nbst:apply-templates/></nbst:otherwise>
  </nbst:choose>
  <nbst:value-of select="$ending"/>
</nbst:template>

<nbst:template match="onepage">
  <nbst:param name="first-time" select="true()"/>
  <nbst:variable name="following" select="following-sibling::onepage">
  <nbst:choose>
    <nbst:when test="$first-time"><nbst:call-template name="tie-number"/></nbst:when>
    <nbst:otherwise><nbst:value-of select="."/></nbst:otherwise>
  </nbst:choose>
  <nbst:if test="$following">
    <nbst:text>, </nbst:text>
    <nbst:apply-templates select="$following[1]">
      <nbst:with-param name="first-time" select="false()"/>
    </nbst:apply-templates>
  </nbst:if>
</nbst:template>

<nbst:template match="firstpage | pages-verbatim">   <!--  Putting a non-breaking space character    -->
  <nbst:call-template name="tie-number"/>            <!--  before a small number.                   -->
</nbst:template>

<nbst:template match="ff">
  <nbst:text> \bblff</nbst:text>
</nbst:template>
```

Figure 5: Putting page numbers down in nbst.

PAGES field. Such XML trees are conformant to a DTD[10], an excerpt from which being given in Figure 3. Syntactically, the PAGES field of MlBibTEX allows the specification of:

- a single page: {353},
- a range of pages: {353--457},
- the first page of an unspecified number of consecutive ones: {353+},
- some enumerated pages: {353,439,519},
- otherwise, the value associated with this field is kept *verbatim* and becomes the content of the pages-verbatim element: this content will appear as it is within any predefined bibliography style.

The bibliography styles of BibTEX deal with these different syntactic forms, as it can be seen in Figure 4, but this style of programming seems to us to be some *hack*.

Figure 5 shows how page numbers can be processed using nbst. Many tags and attributes are the same than in XSLT, except for the namespace used as a prefix, which is obviously different. We explain

[10]**D**ocument **T**ype **D**efinition. A DTD defines a document markup model [29, Ch. 5]. The DTD we use is a revised version of what is given in [10].

Entity reference	Character	How to produce it in LaTeX	Numeric entity
`&amp;`	&	`\&`	`&#38;`
`&apos;`	’		`&#39;`
`&emdash;`	—	`---`	`&#151;`
`&endash;`	–	`--`	`&#150;`
`&eol;`	¶[a]	`\newline`	`&#10;`
`&gt;`	<		`&#62;`
`&lt;`	>		`&#60;`
`&nobsp;`[b]		`~`	`&#160;`
`&quot;`	"		`&#34;`

[a]‘¶’ is a typographic sign for the end-of-line character [2, § 2.85]. In nbst, this entity is used to begin a new line within generated files.

[b]Non-breaking space character.

Table 1: Entities usable in nbst.

later what the parameters `beginning` and `ending` are precisely, but intuitively, we can guess that they are strings to be put before and after the page numbers. Let us notice the use of *variables* — names that may be bound to values — and of *path expressions* in `match` and `select` attributes' values. Using the `following-sibling` axis allows us to reach the subtrees at the right of the current node and sharing the same parent node, that is particularly useful to implement loops, in the sense of ‘classical’ programming languages. Putting some enumerated pages would be done this way if we express it using a ‘classical’ algorithm:

```
write(tie-number(first(one-page-elements))) ;
loop
  one-page-elements ← rest(one-page-elements) ;
  exit when one-page-elements = ∅ ;
  write(",␣") ; write(first(one-page-elements)) ;
end loop ;
```

Figure 5 shows how this algorithm is put into action by means of a recursive template, matching the first element of page numbers not written yet. This technique is very common in XSLT for iterative algorithms.

Let us focus on the texts generated when these templates are invoked, more precisely, on the content of the `nbst:text` tags: we notice the use of additional LaTeX commands, for example, `\bblp` (resp. `\bblpp`) for one (resp. several) pages. These names originate from bibliography styles generated by the makebst program [3] in interface with the babel package [24, Ch. 9], and are language-dependent. For example, the `\bblp` command is expanded in ‘p.’ for ‘page’ in English and French, in ‘S.’ for ‘*Seite*’ in German. How to organise them is shown in [14, § 2].

```
<nbst:template match="lastpage">
  <nbst:value-of
    select="concat('&endash;',.)"/>
</nbst:template>
```

```
<nbst:template match="lastpage"
               language="french">
  <nbst:value-of select="concat('-',.)"/>
</nbst:template>
```

Figure 6: Default and language-dependent templates.

Special characters can be denoted by entity references, like in XML [29, pp. 48–49]. MlBibTEX knows more predefined character entities than XML — e.g., ‘`&endash;`’, used in Figure 6 — they are summarised in Table 1: for each, we give its name, the corresponding character, the way to produce it in LaTeX if this character is special[11], the decimal number coding it w.r.t. Unicode [33].

Now let us introduce the main difference between XSLT and nbst. When a range of pages is to be given, an *en-dash* character[12] should be put between the first and last page numbers. More precisely, this is the convention for most European languages, including English. But in documents written in French, this character tends to be replaced by a single minus character (‘-’). In our style, this character is put by the template processing the last page number. Figure 6 gives two version of this template: a default version, without the `language`, and another version, suitable for the French language. This `language` attribute does not exist in XSLT; in nbst, a template with it has higher priority than the same template without.

Style for a entry type

As we can read in [24, § 13.6.3], introducing small changes in a bibliography style written using the bst language is quite easy. Writing the whole of a style is a worthwhile exercise: we have to know what has been pushed onto the stack handled by BibTEX, what we can pop from it, possibly after applying the `duplicate$` function when this value is needed afterwards by the program. This language is not modular, we have to take care of such questions from a

[11]MlBibTEX uses it only when the `mode` attribute of the `nbst:output` element (cf. Figure 12) is `LaTeX`. For example, the element:

```
<nbst:text>The Bull &amp; the Spear</nbst:text>
```

produces ‘`The Bull \& the Spear`’ (resp. ‘`The Bull & the Spear`’) if the mode is `LaTeX` (resp. `text`).

[12]That is, a dash as wide as the ‘n’ letter.

```
<inproceedings> ::=
    "\bibitem{" <id> "}¶" <authors> <title> <in-eds-booktitle> [", " <volume-number-series>]
    [", " <pages>] <date-etc> ["¶\newblock " <note> "."] "¶¶" ;
<authors>               ::= <name-list> ".¶\newblock " ;
<editors>               ::= <name-list> ", \bbled, "                  if |<name-list>| = 1 |
                            <name-list> ", \bbleds, "                 if |<name-list>| > 1 ;
<name-list>             ::= <name> {", " <name>} [", \bbland\ " <name> | " \bbletal"] ;
<title>                 ::= change-case(t)(<string>) ".¶\newblock " ;
<booktitle>             ::= "\emph{" <string> "}" ;
<in-eds-booktitle>      ::= "\capitalize\bblin " [<editors>] <booktitle> ;
<volume-number-series>  ::= "\bblvol" <tie-number>_<volume> " \bblof \emph{" <series> "}" |
                            "\bblno" <tie-number>_<number> " \bblin " <series> ;
<pages>                 ::= "\bblp" <tie-number(s)>                   if |<tie-number(s)>| = 1 |
                            "\bblpp" <tie-number(s)>                  if |<tie-number(s)>| > 1 ;
<tie-number(s)>         ::= <non-breaking-space-character> <number(s)>   if <number(s)> < 3 |
                            " " <number(s)>                           if <number(s)> ≥ 3 ;
<date-etc>              ::= [", " <address> ", "] <date> [". " <org-pub>] ". " |
                            [". " <org-pub>] ", " <date>
<org-pub>               ::= [<organisation> ", "] <publisher> ;
```

'|...|' is for the number of elements of a list, '$\overline{\ldots}$' for the length of a string. Cf. Table 1 about the '¶' sign.

Figure 7: How to put information about a story included into an anthology.

function to another, and the use of only global variables reinforces this monolithic way of programming. So, the best method for rewriting a style wholly is to express it using a grammar, according to a *reverse engineering*[13] approach. That is, studying bst styles in order to deduce such a grammar. Of course, such modelling can also be done from documents giving rules for bibliographies' layout, such as [1, § 10] or [2, §§ 15 & 16].

Figure 7 gives all the possible texts for references generated by BibTEX, using a 'plain' style and derived from entries being `@INPROCEEDINGS` type. We do not consider cross-referencing ([22, § B.1.4], [24, § 13.2.5]), not implemented yet in MlBibTEX. These possible texts are expressed with a formalism close to EBNF[14], that is:

- for each non-terminal symbol, enclosed like an XML tag, the expression following the '`::=`' sign and terminated by '`;`' states how it can be expanded;
- the '`|`' sign means an alternative, '`[...]`' is for an optional part, '`{...}`' for zero or more occurrences of its content;
- expressions enclosed by two double quote characters are texts to be put: let us recall that they are part of LATEX input.

Since this grammar does not model texts to be parsed, but texts to be generated, we do not have to be conformant with conditions related to parsing, as that would be the case for a language to be interpreted or compiled. In fact, most of our non-terminal symbols are fields' names of MlBibTEX (e.g., `<title>`) or simple types (e.g., `<string>`). There is some language abuse — for example, the use of functions (e.g., `change-case`[15]) — but we think that such a specification is precise and gives a good overview of the texts to be generated.

So, we are given precise information about the order in which fields' values should be placed. As specified in the file plain.bst, we keep the occurrences of the `\newblock` command, used when the bibliography is to be 'open' — by means of the openbib option of the `\documentclass` command — that is, each block starting on a new line [24, § 12.2.1]. On another point, some keywords, hard-wired in this file, are replaced by multilingual commands of LATEX. By the way, let us remark that we are able

[13] According to the terminology used in Software Enginnering:

- **re-engineering** consists of transforming a program written using an 'old' language into a new program in a more modern language: for example, deriving a C program from source files written in FORTRAN;
- **reverse engineering** is the process of analysing software in order to recover its design of specification.

As stated in [31, Ch. 34], reverse engineering is part of software re-engineering process, in the sense that allows better understanding of a system.

[14] **E**xtended **B**ackus-**N**aur **F**orm. Readers unfamiliar with this formalism can refer to [4] for an introduction. DTD syntax originate from it.

[15] Analogous to the namesake function in BibTEX [25].

```
<nbst:template match="inproceedings">
  <nbst:call-template name="common-pre"/>
  <nbst:variable name="comma-space"
                 select="', '"/>
  <nbst:apply-templates select="author"/>
  <nbst:apply-templates select="title"
                        mode="inproc"/>
  <nbst:call-template name="in-eds-booktitle"/>
  <nbst:call-template
    name="volume-number-series">
    <nbst:with-param name="beginning"
                     select="$comma-space"/>
  </nbst:call-template>
  <nbst:variable name="pages">
    <nbst:apply-templates select="pages">
      <nbst:with-param name="beginning"
                       select="$comma-space"/>
    </nbst:apply-templates>
  </nbst:variable>
  <nbst:call-template name="date-etc">
    <nbst:with-param name="previous"
                     select="$pages"/>
  </nbst:call-template>
  <nbst:apply-templates select="note">
    <nbst:with-param
      name="beginning"
      select="'&eol;\newblock '"/>
    <nbst:with-param name="ending"
                     select="'.'"/>
  </nbst:apply-templates>
  <nbst:call-templates name="common-post"/>
</nbst:template>
```

Figure 8: Building a reference from an `inproceedings` element: program using nbst.

to capitalise the result of such a command when it begins a sentence, by means of the `\capitalize` command[16]. As far as possible, we consider that a sign of ponctuation terminates the written form of a field — for example, the list of authors, ended with a period — but it is not always possible: as another example, the specification of page numbers may be followed by a comma if there is an address, by a period if there is an organisation name. In such a case, the sign of ponctuation is specified before the non-terminal symbol it opens in Figure 7.

[16]This command is not predefined in LaTeX, it can be defined as follows:

```
\def\capitalize#1{%
 \def\Capitalize##1{\uppercase{##1}}%
 \expandafter\Capitalize#1}
```

cf. [21] for more details about `\expandafter` and the definitions of TeX commands.

```
<nbst:template match="title" mode="inproc">
  <nbst:apply-templates match=".">
    <nbst:with-param name="emf"
                     select="false()"/>
    <nbst:with-param name="retain-capitals"
                     select="false()"/>
  </nbst:apply-templates>
</nbst:template>
```

Figure 9: Putting titles down.

Now the role of the two template parameters `beginning` and `ending`, occurring in Figure 5 is explained. Their use is systematic, as it can be seen in Figure 8, that 'implements' our specification. More generally, we can notice that writing this template matching `inproceedings` elements is direct, once we got a grammar for such references. If we consider Figure 7, the layout for an element (e.g., `<author>`) is implemented by a template with a `match` attribute; if we implement a non-terminal symbol grouping the layout of several elements (e.g., `<in-eds-booktitle>`), a named template does that. The named template `common-pre` opens a reference, by putting the `\bibitem` command [24, § 12.1.2], whereas the `common-post` template closes it. Both may used to insert multilingual directives, for example, the `otherlanguage` environment of the babel package [24, § 9.2.1].

Let us mention a last point about signs of ponctuation: several consecutive ones may conflict. In practice, such a case occurs when a period is to be put after a string ending with an exclamation or question mark, or with a period belonging to an abbreviation. BibTeX solves this case by means of its function `add.period$` [25], provided that the string has not been popped yet. In XSLT and nbst, a string is output by means of the `value-of` element, unless it is processed within a template that becomes the content of a variable. Thereby the result of this template can be memoized and reused later. Let us look at Figure 8: the string result of invoking the template matching the `pages` element becomes the value of the `pages` variable, which is passed to the named templates `date-etc`.

Refining the way to process `title` elements, let us remark that it depends on the entry type: within the bibliography style plain.nbst, they are put down using italic characters for an entry type being type `@BOOK`, written using roman characters without quotation marks if this type is `@INPROCEEDINGS`. In this last case, we process such an element with a

```
<nbst:template match="title">
  <nbst:param name="emf" select="true()"/>
  <nbst:param name="quotedbf" select="false()"/>
  <nbst:param name="retain-capitals" select="true()"/>
  <nbst:param name="ending" select="'.&eol;\newblock'"/>
  <nbst:if test="$quotedbf"><nbst:text>\begin{bblquotedtitle}</nbst:text></nbst:if>
  <nbst:if test="$emf"><nbst:text>\emph{</nbst:text></nbst:if>
  <nbst:variable name="title-put">
    <nbst;choose>
      <nbst:when test="$retain-capitals"><nbst:apply-templates/></nbst:when>
      <nbst:otherwise>
        <nbst:apply-templates select="node()[1]">
          <nbst:with-param name="retain-capitals" select="false()"/>
          <nbst:with-param name="no-left-lowercase" select="true()"/>
        </nbst:apply-templates>
        <nbst:apply-templates select="node()[position() &gt; 1]">
          <nbst:with-param name="retain-capitals" select="false()"/>
        </nbst:apply-templates>
      </nbst:otherwise>
    </nbst:choose>
  </nbst:variable>
  <nbst:value-of select="$title-put"/>
  <nbst:if test="$emf"><nbst:text>}</nbst:text></nbst:if>
  <nbst:if test="$quotedbf"><nbst:text>\end{bblquotedtitle}</nbst:text></nbst:if>
  <nbst:call-template name="adjoin-sign">
    <nbst:with-param name="the-string" select="$title-put"/>
    <nbst:with-param name="ending" select="$ending"/>
  </nbst:call-template>
</nbst:template>
```

Figure 10: Putting titles down (*continued*).

`mode` attribute, as shown in Figure 9. The template matching `title` elements without any mode — cf. Figure 10 — allows us to define parameters for ruling the layout and give them default values used when we display the title of a book:

- `emf`: if true, use italic characters;
- `quotedbf`: if true, use language-dependent quotation marks, provided by the `bblquotedtitle` environment (cf. [14, § 2]);
- `retain-capitals`: if false, converting the title to lowercase except at the beginning;
- `ending`: the string to be put after the title. The named template `adjoin-sign` prevents conflict between the last character of the title and the value of `ending`.

As shown in Figure 9, this template with the `mode` attribute set to `inproc` only consists of passing suitable values to the general template of Figure 10. Processing titles according to this `inproc` mode can be redefined for the French language, using French quotation marks, or the German language, using italic characters, as written in Figure 11.

Core of a style

When MlBIBTEX builds an XML-like tree with all the entries to be processed, this tree is rooted by an element so-called `mlbiblio`. Figure 12 gives the root element of our 'new plain' bibliography style and shows how to process all the entries. Opening the `thebibliography` environment [24, § 12.1.2] is done by the named template `put-preamble`, which may put additional LATEX definitions, especially those included in BIBTEX's preambles [24, § 13.2.4]. Symmetrically, the `putpostamble` template closes the bibliography.

We can also see how entries are sorted before they are dispatched according to their type. Like the namesake element of XSLT, the first occurrence specifies the primary sort key, the second occurrence the secondary sort key, used for elements left unsorted, and so on. The first occurrence could have been specified by:

```
select="author/name[1]/personname/last"
```

that is, sorting entries w.r.t. the family name of the first author, but that would discard organisation

```
<nbst:template match="title" mode="inproc"
               language="french">
  <nbst:apply-templates match=".">
    <nbst:with-param name="emf"
                     select="false()"/>
    <nbst:with-param name="quotedbf"
                     select="true()"/>
  </nbst:apply-templates>
</nbst:template>

<nbst:template match="title" mode="inproc"
               language="german">
  <nbst:apply-templates match=".">
    <nbst:with-param
      name="ending"
      select="';&eol;\newblock'"/>
  </nbst:apply-templates>
</nbst:template>
```

Figure 11: Putting titles down w.r.t. French and German styles.

names as authors. The solution we put in Figure 12 consists of concatenating three strings related to the first author, two of them being always empty:

- the family name, if this name is for a person,
- the sort key of an organisation name, if given,
- the organisation name itself, if the sort key has not been given.

For first authors that are organisation names, only the first occurrence of the `nbst:sort` element is of interest, the others do nothing. When sorting entries w.r.t. names is finished, we sort w.r.t. years, then months. This last sort order can seem to be some *hack* since it uses the interface with Scheme functions (cf. § B), but let us recall that programming such a sort order is very difficult in bst and unused in practice. However, we think that our successive `nbst:sort` elements are clearer than the `presort`, `sortify` and `purify$` functions used within bibliography styles written in bst.

Splitting a style into several files

As abovementioned, the bst language is not modular, and all the definitions for a particular style must be stored in the same file, what is a drawback since several styles share the same definitions. That complicates the mainenance of bibliography styles if some definitions need some enrichment. Besides, it is difficult, when we are studying a style, to determine what is specific or common to other styles. The nbst language includes:

- an `nbst:include` element, to import definitions explicitly from another nbst file;
- *implicit importations*, described in [14, § 3.1].

Hereafter, we show how to spread out the templates we are writing over different files, in order to take as much advantage as possible of implicit importations of nbst. Let us recall that we are developing a new version of the 'plain' bibliography style, that is, the master file is plain.nbst.

- The global.nbst can be viewed as MlBIBTEX's initial library of definitions using nbst: it includes general named templates such as:

  `adjoin-sign`   `date-etc`   `tie-number`

  as well as template matching the following elements:

`address`	`one-page`
`booktitle`	`orgnization`
`ff`	`pages`
`firstpage`	`pages-verbatim`
`lastpage`	`publisher`
`note`	`title`

  Putting more templates in this file may seem to be of interest, but let us recall that in nbst, imported templates have the same priority than other elements[17]: so 'global' elements cannot be redefined[18], unless adding a `language` or `mode` attribute to the redefinition.
- Of course, the plain.nbst file — the master file for this bibliography style — must include its root (`nbst:bst`) element and the 'main' template matching an `mlbiblio` element, given in Figure 12. The following named templates, related to references' labels, should be included in this file, too:

`common-post`	`put-postamble`
`common-pre`	`put-preamble`

  The layout of the following element depends on the bibliography style, so the corresponding templates have to be stored in the plain.nbst file:

`author`	`inproceedings`	`series`
`editor`	`number`	`volume`

  as well as the named templates, for the same reason:

  `in-eds-booktitle` `volume-number-series`
  `org-pub`
- The 'French' definition of the template matching a `lastpage` element (cf. Figure 6) is general for French-speaking styles, not directly related to 'plain' styles, so we place it onto the

[17]This is not the case in XSLT if the `xsl:import` element is used.

[18]More exactly, if there is conflict between templates, it is unpredictible to know which will be run.

```
<nbst:bst version="1.3" id="plain" xmlns:nbst="http://lifc.univ-fcomte.fr/~hufflen/mlbibtex">

  <nbst:output method="LaTeX" encoding="ISO-8859-1"/>
  <!--  This encoding allows accented letters of Western European Languages [5, Table C.4].     -->

  <nbst:template match="mlbiblio">
    <nbst:call-template name="put-preamble">
      <nbst:with-param name="longest-label" select="count(*)"/>
    </nbst:call-template>
    <nbst:apply-templates>
      <nbst:sort select="concat(author/name[1]/personname/last,
                                author/name[1]/othername/@sortingkey,
                                author/name[1]/othername[not(@sortingkey)])"/>
      <nbst:sort select="author/name[1]/personname/first"/>
      <nbst:sort select="author/name[1]/personname/von"/>
      <nbst:sort select="author/name[1]/personname/junior"/>
      <nbst:sort select="year" data-type="number"/>
      <nbst:sort select="call(month-position,month)" data-type="number"/>
    </nbst:apply-templates>
    <nbst:call-template name="put-postamble"/>
  </nbst:template>

  ...

</nbst:bst>
```

Figure 12: Root element for a program in nbst — Organising all the entries to generate references.

-french.nbst file, that is, the file grouping the general definitions for the French language.

- On the contrary, the French and German redefinitions of the template matching `title` elements in `inproc` mode (cf. Figure 11) belong both to the 'plain' bibliography style so they should be put into the files plain-french.nbst and plain-german.nbst.

Conclusion

We think that when a new tool or a new programming language is developed, its conceptor(s) should provide methodology and advice about it. Often teachers of programming languages notice that students may program badly in a good language. Let us go back to BIBTEX, we personally missed — in the past, a long time before we decided to develop MlBIBTEX — a didactic introduction to the bst language like [28]. Likewise, an overview for writers of LATEX extensions such as [24, Appendix A] was missing for a long time.

In this article, we have not shown all the features of MlBIBTEX. For example, we have not gone thoroughly into multilingual features — in order to show that our approach was mostly suitable for designing styles using XSLT, too — and 'new plain' style was implicitly supposed to be language-dependent [13], that is, each reference is expressed using the language's entry. In fact, our goal was to show that nbst allowed us to write bibliography styles in elegant manner, provided that we are given a precise specification of what to put. So we are able to build a solid basis for a style, and people could easily enrich it with new language-dependent templates by using MlBIBTEX's implicit importation.

Now we are rewriting predefined bibliography styles of BIBTEX. Most of them have already been redesigned, but this work is not finished yet at the time we finish writing this article. We hope that these explanations would help people enrich these new styles, especially in order to adapt them to other languages.

Acknowledgements

Thanks to Volker R. W. Schaa, who proof-read the German translation of the abstract.

A bst vs nbst

A precise comparison between bst and nbst is difficult, since these two languages belong to very different programming paradigms. The former is based on handling a stack (see [28] for a didactic introduction to this aspect), the latter encourages rule-based programming. They do not belong to the same time, either: the former has been influenced by assembly

bst expression	"Equivalent" expression in nbst	Kind[a]
$\mathcal{I}_1$ $\mathcal{I}_2$ >	$\mathcal{I}_1^\natural$ &gt; $\mathcal{I}_2^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ <	$\mathcal{I}_1^\natural$ &lt; $\mathcal{I}_2^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ =	$\mathcal{I}_1^\natural$ = $\mathcal{I}_2^\natural$	P
$\mathcal{S}_1$ $\mathcal{S}_2$ =	$\mathcal{S}_1^\natural$ = $\mathcal{S}_2^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ +	$\mathcal{I}_1^\natural$ + $\mathcal{I}_2^\natural$	P
$\mathcal{I}_1$ $\mathcal{I}_2$ -	$\mathcal{S}_1^\natural$ - $\mathcal{S}_2^\natural$	P
$\mathcal{S}_1$ $\mathcal{S}_2$ *	concat($\mathcal{S}_1^\natural$,$\mathcal{S}_2^\natural$)	P

$\mathcal{S}$ add.period$	<nbst:call-template name="adjoin-sign">
	  <nbst:with-param name="the-string" select="$\mathcal{S}^\natural$"/>
	  <nbst:with-param name="ending" select="'.'"/>
	</nbst:call-template>
		E[b]

$\mathcal{S}$ "t" change.case$	concat(substring($\mathcal{S}^\natural$,1,1),lowercase(substring($\mathcal{S}^\natural$,2)))	P[c]
$\mathcal{S}$ "l" change.case$	lowercase($\mathcal{S}^\natural$)	P
$\mathcal{S}$ "u" change.case$	uppercase($\mathcal{S}^\natural$)	P
$\mathcal{S}$ chr.to.int$	(char->integer $\mathcal{S}^\natural$)	S
cite$	@id	P
$\mathcal{L}$ empty$	not(string($\mathcal{L}^\natural$))	P

$\mathcal{I}$ $\mathcal{F}_1$ $\mathcal{F}_2$ if$	<nbst:choose>
	  <nbst:when test="$\mathcal{I}^\natural$ &gt; 0">$\mathcal{F}_1^\natural$</nbst:when>
	  <nbst:otherwise>$\mathcal{F}_2^\natural$</nbst:otherwise>
	</nbst:choose>
		E

$\mathcal{I}$ int.to.chr$	(integer->char $\mathcal{I}^\natural$)	S
$\mathcal{I}$ int.to.str$	string($\mathcal{I}^\natural$)	P
$\mathcal{L}$ missing$	not($\mathcal{L}^\natural$)	P
newline$	<nbst:text>&eol;</nbst:text> or <nbst:value-of select="'&eol;'"/>	E
$\mathcal{S}$ num.names$	count(name) if name($\mathcal{S}^\natural$) $\in$ {author, editor}	P
preamble$	@preamble	P
$\mathcal{S}$ purify$	call(bst-purify,$\mathcal{S}^\natural$)	P[d]
quote$	<nbst:text>&quot;</nbst:text> or <nbst:value-of select="'&quot;'"/>	E
$\mathcal{S}$ $\mathcal{I}_1$ $\mathcal{I}_2$ substring$	substring($\mathcal{S}^\natural$,$\mathcal{I}_1^\natural$,$\mathcal{I}_2^\natural$) if $\mathcal{I}_1 > 0$ substring($\mathcal{S}^\natural$,string-length($\mathcal{S}^\natural$) $+ \mathcal{I}_1^\natural - \mathcal{I}_2^\natural + 2$,$\mathcal{I}_2^\natural$) if $\mathcal{I}_1 < 0$	P[c]
$\mathcal{S}$ text.length$	string-length($\mathcal{S}^\natural$)	P
$\mathcal{S}$ $\mathcal{I}$ text.prefix$	substring($\mathcal{S}^\natural$,1,$\mathcal{I}^\natural$)	P[c]
type$	name()	P
$\mathcal{S}$ warning$	<nbst:warning>$\mathcal{S}^\natural$</nbst:warning>	E
$\mathcal{S}$ width$	(tex-width $\mathcal{S}^\natural$)	S[e]
$\mathcal{S}$ write$	<nbst:value-of select="$\mathcal{S}^\natural$"/>	E

[a]Qualifies the given expression in nbst: 'E' is for 'element', 'P' for 'path expression', 'S' for 'Scheme expression'.
[b]The adjoin-sign is included in MlBibTEX's initial library.
[c]Let us recall that indexing strings is one-based in XPath and nbst, whereas it is zero-based in Scheme.
[d]This Scheme function is given in Figure 13. Useless in practice (see Figure 8)!
[e]Not implemented yet (always returns "0").

Table 2: Translation of most bst statements given in [24, Table 13.8]

languages, the latter has taken advantage of a modern langage, suitable for handling documents and designed for a large purpose.

Some statements of bst are not really translatable into nbst: for example, the assignment (':='), because nbst is like a purely functional language, in the sense that a variable — or a parameter — cannot be changed, once it has been given a value. On the other hand, nbst allows recursive templates, like in XSLT, what is useful for iterative programming (cf. Figure 5) and replaces the `while$` function of bst.

The `call.type$` function of bst does not have a direct equivalent, either: such an operation is performed by pattern-matching by means of the `match` attribute of suitable `nbst:template` elements. The

```
(define (bst-purify string-0)
  (let thru ((index (- (string-length string-0) 1))
             ;; Current index, we are going backward. The second argument allows us to accumulate retained
             ;; characters in a list, we begin with an empty list:
             (accumulator '()))
    (if (negative? index)
        ;; The string has been processed, we convert the list of accumulated characters into a string:
        (list->string accumulator)
        (thru (- index 1) (let ((current-char (string-ref string-0 index)))
                            ;; Discarding it if it is not alphanumeric:
                            (if (or (char-alphabetic? current-char) (char-numeric? current-char))
                                (cons current-char accumulator)
                                accumulator))))))
```

Figure 13: Scheme function implementing the bst function `purify$`.

`format.name$` function is replaced by handling path expressions like in XPath for subtrees for authors and editors.

Table 2 is an attempt to express the relationship between bst statements and corresponding realisations in nbst. In fact, it emphasises which statements are easily translatable, which are not. This table does not include bst functions such as ':=', `while$`, `call.type$`, `skip$`. Likewise, we did not put bst functions directly related to BIBTEX's stack management: `duplicate$`, `stack$`, `swap$`, `top$`.

For the other bst functions, we make precise its operands: $\mathcal{I}$ is for an integer, $\mathcal{S}$ for a string, $\mathcal{L}$ for any value, $\mathcal{F}$ for a function. When several operands are the same type, we use indices. We use the '$\ldots^{\natural}$' notation to mean 'the translation of an operand in nbst': for example, the `if$` function of bst pops three values from the stack, the translation of the first should be used inside the value of a `test` attribute, the others should be translated into nbst elements put as contents of an `nbst:if` element.

As it can be seen in Table 2, the direct translation of some statements needs to call functions directly written in Scheme: we put them for sake of completeness, but in practice, most of these functions are useless when a style is wholly rewritten using nbst (cf. § B). Last, let us remark that in the path expressions given in this table — `@id` and `@preamble` — the current node is supposed to be the node for an entry (`inproceedings`, `book`, ...)

B Interface with Scheme

Path expressions used within nbst include calls to external functions written in Scheme and returning strings. The syntax is:

$$\texttt{call(}\textit{function-name}\texttt{,}\textit{arg}_1\texttt{,...,}\textit{arg}_n\texttt{)}$$

where *function-name* is the function's name, applied to arg_1, ..., arg_n ($n \in \mathbb{N}$). Now we got some experience in writing bibliography styles, and as far as we know, there are three reasons for using such functions within bibliography style files:

- functions related to TEX's features: for example, returning the width of a string, expressed in TEX's units (cf. Table 2), as another example, searching LATEX source files: for instance, we have to do that in order to know the document's language[19];
- operations that would be tedious with the functions of XPath's library: an example appearing in Table 2 is the `bst-purify` function;
- functions used to sort entries: e.g., the function `month-position`, that allows the sort of month names according to the chronological order, used in the template given in Figure 8.

In Figure 13, we give the exact equivalent for the `purify$` function of bst, in order to give some idea about how to deal with strings in Scheme. Let us remark that this operation — used in BIBTEX to build strings to be sorted lexicographically — is useless practically since it is better to use successive `nbst:sort` elements as we show in Figure 12.

In addition to the `bst-purify` function, we give a second example written in Scheme in Figure 14: the `month-position` function, used to sort month names, as shown in Figure 12. As abovementioned, this way may be thought as *ad hoc* method, nevertheless, let us remark that such a sort is not provided by 'old' BIBTEX.

[19] See [16] for more details about this problem. MlBIBTEX also searches auxiliary (.aux) files produced by LATEX, but not whilst a bibliography style is applied.

```
(define month-position
  (let ((month-name-list
         '("jan" "feb" "mar" "apr" "may" "jun" "jul" "aug" "sep" "oct" "nov" "dec")))
    (lambda (string-0)
      (let thru ((month-name-list-0 month-name-list)
                 (current-position 0))
        (if (or (null? month-name-list-0)
                ;; This way, elements with a non-recognised or empty month name will be put after those with
                ;; an actual month name after the sorting operation.
                (string=? (car month-name-list-0) string-0))
            (number->string current-position)   ; Final result as a string.
            (thru (cdr month-name-list-0) (+ current-position 1)))))))
```

Figure 14: Scheme function used to sort month names by sorting corresponding positions.

References

[1] Judith BUTCHER: *Copy-Editing. The Cambridge Handbook for Editors, Authors, Publishers.* 3rd edition. Cambridge University Press. 1992.

[2] *The Chicago Manual of Style.* The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.

[3] Patrick W. DALY: *Customizing Bibliographic Style Files.* Version 3.2. February 1999. Part of BIBTEX's distribution.

[4] Lars Marius GARSHOL: *BNF and EBNF: What Are They and How Do They Work?* July 2003. http://www.garshol.priv.no/download/text/bnf.html.

[5] Michel GOOSSENS and Sebastian RAHTZ, with Eitan M. GURARI, Ross MOORE and Robert S. SUTOR: *The LATEX Web Companion.* Addison-Wesley Longmann, Inc., Reading, Massachusetts. May 1999.

[6] Vidar Bronken GUNDERSEN and Zeger W. HENDRIKSE: *BIBTEX as XML Markup.* January 2003. http://bibtexml.sourceforge.net.

[7] Harald HARDERS: „Mehrsprachige Literaturverzeichnisse: Anwendung und Erweiterung des Pakets babelbib“. *Die TEXnische Komödie*, Bd. 4/2003, S. 39–63. November 2003.

[8] Erik VAN HERWIJNEN: *Practical* SGML. Interpharm Press. December 1994.

[9] Jean-Michel HUFFLEN: “MlBIBTEX: a New Implementation of BIBTEX”. In: *EuroTEX 2001*, p. 74–94. Kerkrade, The Netherlands. September 2001.

[10] Jean-Michel HUFFLEN: “Multilingual Features for Bibliography Programs: From XML to MlBIBTEX”. In: *EuroTEX 2002*, p. 46–59. Bachotek, Poland. April 2002.

[11] Jean-Michel HUFFLEN: “Mixing Two Bibliography Style Languages”. In: LDTA *2003*, Vol. 82.3 of ENTCS. Elsevier, Warsaw, Poland. April 2003.

[12] Jean-Michel HUFFLEN: “European Bibliography Styles and MlBIBTEX”. TUG*boat*, Vol. 24, no. 3. EuroTEX 2003, Brest, France. June 2003.

[13] Jean-Michel HUFFLEN: “MlBIBTEX's Version 1.3”. TUG*boat*, Vol. 24, no. 2, p. 249–262. July 2003.

[14] Jean-Michel HUFFLEN: “Making MlBIBTEX Fit for a Particular Language. Example of the Polish Language”. *Biuletyn* GUST, Vol. 21, p. 14–26. 2004.

[15] Jean-Michel HUFFLEN: “A Tour around MlBIBTEX and Its Implementation(s)”. *Biuletyn* GUST, Vol. 20, p. 21–28. In *BachoTEX 2004 conference.* April 2004.

[16] Jean-Michel HUFFLEN: “MlBIBTEX: beyond LATEX”. In: *International Conference on TEX,* XML*, and Digital Typography*, Vol. 3130 of LNCS, p. 203–215. Springer, Xanthi, Greece. August 2004.

[17] Jean-Michel HUFFLEN: *Multilingual Bibliography Styles:* nbst *vs* XSLT. To appear in Proc. GUIT conference, Pisa. October 2004.

[18] International Standard ISO/IEC 10179:1996(E): DSSSL. 1996.

[19] Richard KELSEY, William D. CLINGER, Jonathan A. REES, Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIG, Daniel P. FRIEDMAN, Robert HALSTEAD, Chris HANSON, Christopher T. HAYNES, Eugene Edmund KOHLBECKER, JR, Donald OXLEY, Kent M. PITMAN, Guillermo J. ROZAS, Guy Lewis STEELE, JR, Gerald Jay SUSSMAN and Mitchell

WAND: *Revised^5 Report on the Algorithmic Language Scheme*. February 1998. http://www.cs.indiana.edu/scheme-repository/.

[20] Oleg KISELYOV: "A Better XML Parser through Functional Programming". In: *4th International Symposium on Practical Aspects of Declarative Languages*, Vol. 2257 of LNCS. Springer. 2002.

[21] Donald Ervin KNUTH: *Computers & Typesetting. Vol. A: the TEXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[22] Leslie LAMPORT: *LATEX. A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[23] Wenzel MATIASKE: *Multilinguale Zitierformate*. Oktober 1995. CTAN:macros/latex/contrib/supported/mlbib/.

[24] Frank MITTELBACH and Michel GOOSSENS, with Joannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The LATEX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[25] Oren PATASHNIK: *Designing BIBTEX Styles*. February 1988. Part of BIBTEX's distribution.

[26] Oren PATASHNIK: *BIBTEXing*. February 1988. Part of BIBTEX's distribution.

[27] Chris PUTNAM: *Bibliography Conversion Utilities*. February 2005. http://www.scripps.edu/~cdputnam/software/bibutils/bibutils.html.

[28] Bernd RAICHLE: *Tutorium: Einführung in die BIBTEX-Programmierung*. Handouts für DANTE 2002. Februar 2002.

[29] Erik T. RAY: *Learning XML*. O'Reilly & Associates, Inc. January 2001.

[30] John E. SIMPSON: *XPath and XPointer*. O'Reilly & Associates, Inc. August 2002.

[31] Ian SOMMERVILLE: *Software Engineering*. 5th edition. Addison-Wesley Publishing Company. 1996.

[32] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.

[33] THE UNICODE CONSORTIUM: *The Unicode Standard Version 4.0*. Addison-Wesley. August 2003.

[34] Doug TIDWELL: XSLT. O'Reilly & Associates, Inc. August 2001.

[35] W3C: *XML Path Language (XPath). Version 1.0*. W3C Recommendation. Edited by James Clark and Steve DeRose. November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[36] W3C: *XSL Transformations (XSLT). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[37] W3C: *Extensible Stylesheet Language (XSL). Version 1.0*. W3C Recommendation. Edited by James Clark. October 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/.

[38] Norman WALSH and Leonard MUELLNER: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.

"La machine à formulaires" (the forms' machine), or TEX for a Kafkaian world

Antoine Lejay

November 29, 2004

Abstract

This article describes the *Machine à formulaires*, whose goal is to fill in different forms from a single file. Its aim is to help candidates to positions of (assistant) professors in a French University not to lose time in copying various informations regarding each position they apply.

1 Introduction

This article contains a short description of a set of TEX files designed to help candidates to professors' or assistant professors' positions in French Universities to fill in heavy forms. In some sense, it is similar to the AMS cover initiative [1], but with a different philosophy and design, and where informations specific to each position are reported in each form.

The *Machine à formulaires* (the forms' machine, or MAF in short) has been made available since 2001 on the WEB sites of the *Opération Postes* [5] and the *Guilde des Doctorants* [2][1] whose goal is to help Ph.D. holders looking for a job. It seems to have been appreciated by numerous candidates, since it allows them to gain a lot of time in the constitution of their application forms. The MAF does not need any real knowledge in TEX/LATEX (only common sense) and shows the ability of TEX to deal with tasks such as automatic creation of forms without using exterior softwares.

Basically, all the informations are entered in a single file with a rather natural syntax (provided by the `keyval` package) and any repetition is avoided.

[1]A package with the same goal have been already available on that site and the MAF represents an attempt to have a more flexible way to do the job.

Various forms may be produced simply by changing the class file which is loaded. Of course, data and presentation commands are completely separated.

2 The difficulty of applying to a French university...

The procedure for applying to a French university as *maître de conférences* (assistant professor) or professor relies on a national-wide, official and strict process. Almost all the French universities are public and then (assistant) professors are state-employees. In automn and winter, there are discussions between each university and the French Department of Education to obtain the creation of new positions, or the replacement of people leaving or retiring. Then all the available positions, whatever the university and the domain, are published around February of each year in the *Journal Officiel de la République Française* (also called the “JO”), which is a daily publication for the new legal or official texts (laws, decrees, ...) of the French government. In the JO, a range of dates are given for: applying for a position (for the candidate), choosing the candidates the university wishes to hire and sending their name to the French Department of Education. The JO also specifies what the applicants send to the universities.

Basicall, to each position is assigned: (a) an identification number; (b) a university; (c) one or many research domains. Generally, the lab proposing this position may be deduced from this information, but this is not always true. The research domains are classified, each one corresponding to a subgroup of the *Conseil National des Universités* (CNU). Each of these subgroups is labelled by a number (for example, 25 for Pure Mathematics, 26 for Applied Mathematics, 27 for Computer Sciences, ...). Every French Ph.D. thesis is also classified according to these domains (but a Ph.D. holder is not restricted to apply to positions only in his domain); (d) possibly some precisions about the (research or teaching) skills that are required (The position may be for a team within a lab, but it may also happen that all the teams of a lab have to choose together who to hire); (e) possibly other legal informations that would be too long to explain here. Yet it concerns only a few positions each year.

For example, one could read in the JO containing some lists of positions looking like

> POSTES DE MAÎTRES DE CONFÉRENCES
> ...
> *26^e section : mathématiques appliquées et applications des mathématiques*
> Université Grenoble-I : et 27^{e} section, bio-informatique : 1445.
> Université de Pau : 0706.
> ...

The first position — identified by the number 1445 — is available at one of the two universities of the city of Grenoble. It is primarily for people working in bio-informatic, either with a Ph.D. in Computer Science or Applied Mathematics. The applicant have to contact the University Grenoble I if she/he want to known which lab or team is concerned by this position. The second position, at University of Pau, is for an applied mathematician whatever his/her speciality. Of course, the priority will generally be given to the candidates that may interact with the people there.

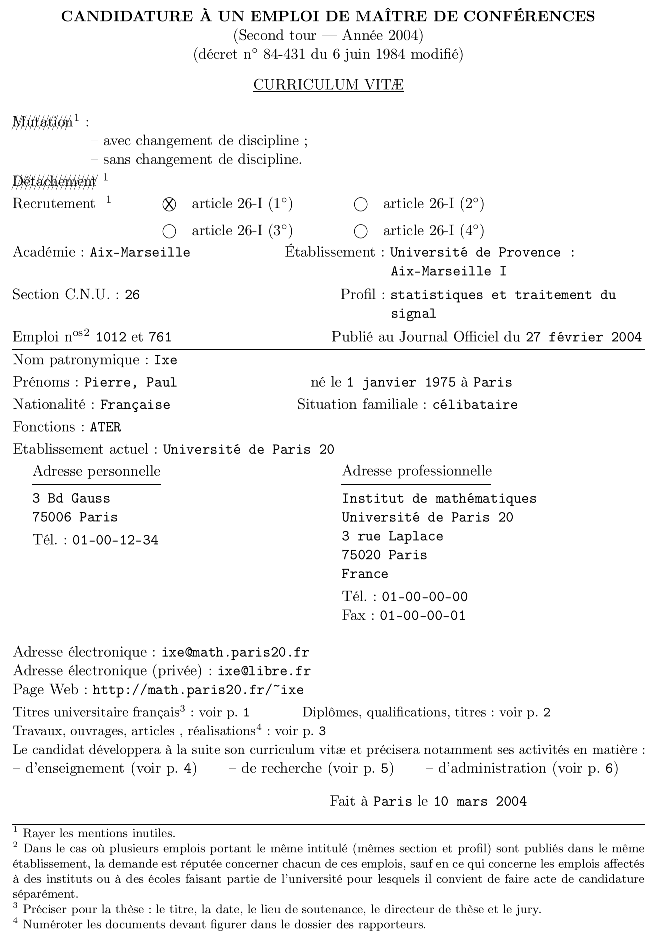
Once having read the JO, the candidate willing to apply for some position shall send to the university offering it two copies of: (a) a normalized form (nicknamed[2] *Annexe B* by the candidates, since its presentation appears each year in the JO in the Section *Appendix B*) in which he states he applies to this position, with personal informations and all the informations given above (identification number, ...); (b) a curriculum vitæ whose first page (nicknamed *Annexe C*, since its appears each year in the Section *Appendix C*) also follows a normalized presentation with again these informations. These two forms may be found in Figure 1

Since a candidate generally send 10, 20 or more application forms that are all different, filling them within a few weeks takes a lot of time, increases the natural stress of the candidate (it is not easy to get a job) and looks like an administrative nightmare.

3 Goals and design

Of course, in days where everybody has access to a computer, the candidate may wish to produce automatically these forms. A truly helpful code shall in my opinion follow the following specifications: (a) the syntax shall be simple and rather natural; (b) data and presentation shall be separated; (c) replication of information shall be limited to reduce the risk of errors; (d) it must be system independent to be available to the maximum amount of people; (e) extensions must be easy to write in order to produce envelops labels, cover letters,...

[2]It is interesting to note how an administration tends to create its own jargon...



CANDIDATURE À UN EMPLOI DE MAÎTRE DE CONFÉRENCES
(Second tour — Année 2004)
(décret n° 84-431 du 6 juin 1984 modifié)

CURRICULUM VITÆ

~~Mutation~~[1] :
– avec changement de discipline ;
– sans changement de discipline.

~~Détachement~~ [1]

Recrutement [1] ⊗ article 26-I (1°) ○ article 26-I (2°)
○ article 26-I (3°) ○ article 26-I (4°)

Académie : `Aix-Marseille` Établissement : `Université de Provence : Aix-Marseille I`

Section C.N.U. : `26` Profil : `statistiques et traitement du signal`

Emploi n^{os}[2] `1012 et 761` Publié au Journal Officiel du `27 février 2004`

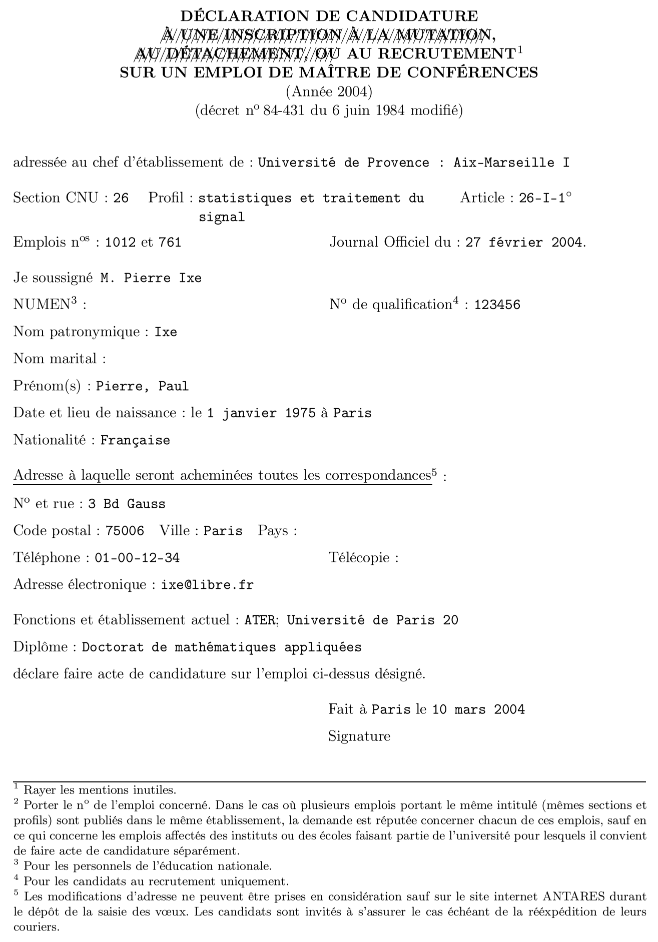
Nom patronymique : `Ixe`
Prénoms : `Pierre, Paul` né le `1 janvier 1975` à `Paris`
Nationalité : `Française` Situation familiale : `célibataire`
Fonctions : `ATER`
Etablissement actuel : `Université de Paris 20`

Adresse personnelle	Adresse professionnelle
3 Bd Gauss 75006 Paris Tél. : 01-00-12-34	Institut de mathématiques Université de Paris 20 3 rue Laplace 75020 Paris France Tél. : 01-00-00-00 Fax : 01-00-00-01

Adresse électronique : `ixe@math.paris20.fr`
Adresse électronique (privée) : `ixe@libre.fr`
Page Web : `http://math.paris20.fr/~ixe`
Titres universitaire français[3] : voir p. 1 Diplômes, qualifications, titres : voir p. 2
Travaux, ouvrages, articles , réalisations[4] : voir p. 3
Le candidat développera à la suite son curriculum vitæ et précisera notamment ses activités en matière :
– d'enseignement (voir p. 4) – de recherche (voir p. 5) – d'administration (voir p. 6)

Fait à `Paris` le `10 mars 2004`

[1] Rayer les mentions inutiles.
[2] Dans le cas où plusieurs emplois portant le même intitulé (mêmes section et profil) sont publiés dans le même établissement, la demande est réputée concerner chacun de ces emplois, sauf en ce qui concerne les emplois affectés à des instituts ou à des écoles faisant partie de l'université pour lesquels il convient de faire acte de candidature séparément.
[3] Préciser pour la thèse : le titre, la date, le lieu de soutenance, le directeur de thèse et le jury.
[4] Numéroter les documents devant figurer dans le dossier des rapporteurs.



DÉCLARATION DE CANDIDATURE
~~À UNE INSCRIPTION À LA MUTATION~~,
~~AU DÉTACHEMENT, OU~~ AU RECRUTEMENT[1]
SUR UN EMPLOI DE MAÎTRE DE CONFÉRENCES
(Année 2004)
(décret n° 84-431 du 6 juin 1984 modifié)

adressée au chef d'établissement de : `Université de Provence : Aix-Marseille I`

Section CNU : `26` Profil : `statistiques et traitement du signal` Article : `26-I-1°`

Emplois n^{os} : `1012 et 761` Journal Officiel du : `27 février 2004`.

Je soussigné `M. Pierre Ixe`
NUMEN[3] : N° de qualification[4] : `123456`
Nom patronymique : `Ixe`
Nom marital :
Prénom(s) : `Pierre, Paul`
Date et lieu de naissance : le `1 janvier 1975` à `Paris`
Nationalité : `Française`

Adresse à laquelle seront acheminées toutes les correspondances[5] :
N° et rue : `3 Bd Gauss`
Code postal : `75006` Ville : `Paris` Pays :
Téléphone : `01-00-12-34` Télécopie :
Adresse électronique : `ixe@libre.fr`

Fonctions et établissement actuel : `ATER; Université de Paris 20`
Diplôme : `Doctorat de mathématiques appliquées`
déclare faire acte de candidature sur l'emploi ci-dessus désigné.

Fait à `Paris` le `10 mars 2004`
Signature

[1] Rayer les mentions inutiles.
[2] Porter le n° de l'emploi concerné. Dans le cas où plusieurs emplois portant le même intitulé (mêmes sections et profils) sont publiés dans le même établissement, la demande est réputée concerner chacun de ces emplois, sauf en ce qui concerne les emplois affectés des instituts ou des écoles faisant partie de l'université pour lesquels il convient de faire acte de candidature séparément.
[3] Pour les personnels de l'éducation nationale.
[4] Pour les candidats au recrutement uniquement.
[5] Les modifications d'adresse ne peuvent être prises en considération sauf sur le site internet ANTARES durant le dépôt de la saisie des vœux. Les candidats sont invités à s'assurer le cas échéant de la réexpédition de leurs couriers.

Figure 1: The two forms *Annexe B* and *Annexe C*.

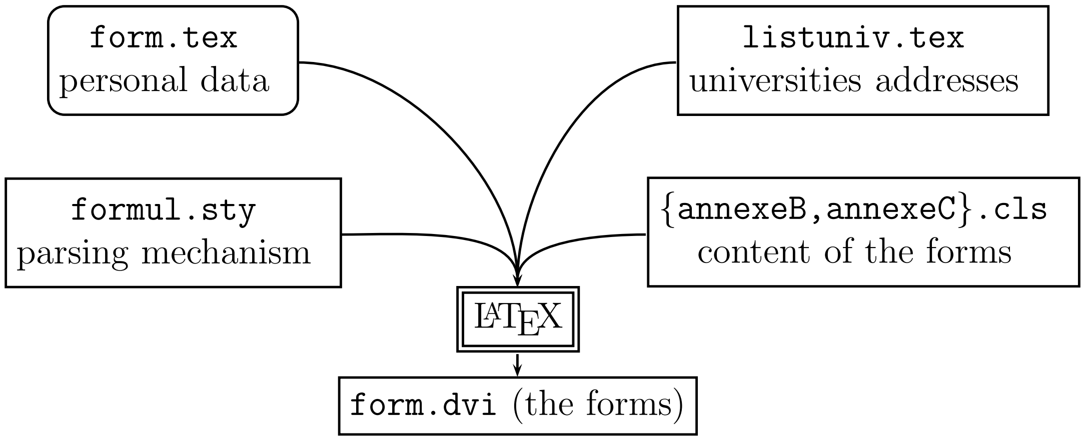


Figure 2: The design of the MAF.

Although many word-processors allow to do such a thing, TEX is probably the best candidate to satisfy point (d) while it does not take too much time to write some code satisfying the other requirements.

The design of the MAF is the following: (a) The candidate enters in a single file (`form.tex`) both personal informations (name, address, ...) and the list of positions she/he plans to apply. We use the functionalities of the D. Carliste's `keyval` package [3] in order to have a simple and clear syntax. (b) The class used in `form.tex` specifies the form to be produced: a single compilation of `form.tex` with the class `annexeC.cls` produces all the first pages of the curriculum vitæ for all the positions. (c) All the class files call the style file `formul.sty` whose goal is to parse `form.tex`, while the class file is devoted to the presentation of the form. Thus, it is rather easy to write or change a class file, while `formul.sty` contains more complex code (not to speak of `keyval.sty`). (d) The complete names and addresses of the universities are available in the separate file `listuniv.tex`, so that the candidate does not lose time in entering these informations.

As such, there are three layers: the front-end user, even if a complete beginner, has only to enter the informations in `form.tex`. If she/he is not satisfied by the output (which tries to be faithful to the given models, but which is not aesthetic), an average LaTeX user may rather easily change the class files or create new ones simply by playing with boxes, spaces and springs, since the parsing and sometimes tricky job is done in `formul.sty`[3].

[3]This style file also provides also some macros for formatting lists and other typesetting issues.

4 The structure of the files

The end-user has only to change or complete the file `form.tex` that begins with

```
\documentclass{annexeB}
%\documentclass{annexeC}
\input{listeuniv}
\begin{document}
```

Here, the user chooses which form to produce by commenting and decommenting the first lines. Then, she/he enters some personal informations (name, address, ...), using key-values syntax.

```
\candidat{
MMmeMlle={M},
nom={Doe},
prenom={John},
...}
```

In order to avoid repetitions (and mistakes), some default informations can be entered with the command `\postedefaut`. For example, a Ph.D. holder in pure mathematics generally applies to positions in the research domain labelled by the number 25. Thus, there is no need to repeat this information, which is then entered as a value for the key `sect`. The key `mdcouprof` accepts `mdc` (for positions of assistant professor) or `prof` (for positions of professor).

```
\postedefaut{
mdcouprof={mdc},
sect={25},
type={recrutement},
dateJO={27 f\'evrier 2004},
articleJO={26I-1}
}
```

Here, the other informations specify in some sense the statut the applicant: for example, the recruitment procedure is also valid for people already having a job as (assistant) professor but willing to go elsewhere. It may also happen that some positions, although rare, are opened only to a restricted category of people.

The file ends by a list of commands `\postes`, one for each position, whith two mandatory arguments (the number of the position and the "tag-name" of the university. This "tag-name" is defined in the file `listuniv.tex` but it is easily deduced from the real name of the university). Some optional arguments, again with the key-value syntax, may be used to give more precision about the position or to override the arguments used by default.

```
\poste[
sect={26},
profil={statistiques}
]{1012}{aix-marseille1}
```

The file `listuniv.tex` contains the list of all the Universities with their addresses, under the form (this list, with 91 entries, was established from the informations given in the WEB site of the french Department of Education [4]):

```
\defuniv{aix-marseille1}{
nom={Universit\'e de Provence~: Aix-Marseille I},
academie={\dapostrophe Aix-Marseille},
adresse={3, place Victor Hugo\\
13331 MARSEILLE CEDEX 3 }
}
```

The command `\dapostrophe` may be used to write "d'Aix-Marseille" (literally "from Aix-Marseille") or "Aix-Marseille" in function of the context. Thus, in the last call of the command `\poste` above, the tag-name `aix-marseille1` means that the full name, address, ... of the corresponding entry in `listuniv.tex` will be used. These data are stored in TEX's memory using a `\csname`/`\endcsname`'s scheme.

When compiling the file `form.tex` with LATEX, the appropriate class file will be called. Every class file shall have the following structure:

```
\LoadClass[11pt]{article}
\RequirePackage{formul}
\newcommand{\codeposte}{
...
}
```

The package `formul.sty` contains the parser. The informations on the candidate are transformed into some commands. For example, the name of the candidates is defined as the expansion of the command `\nom`, ... The command `\codeposte` is called each time the command `\poste` is encountered in the file `form.tex`. Its effect is to typeset and fill the form with the specific informations on the position. The file `formul.sty` also contains some commands regarding typesetting (lists manipulations, ...), so that new class files can be easily created.

5 Conclusion

The first version was released in 2001 and its seems that many candidates enjoyed it (I have absolutely no statistics at all, but I have received some very enthusiastic emails), since it allows them to concentrate on the content of the curriculum vitæ and not on writing boring, repetitive informations.

Using TEX for this task is advantageous since only one version needs to be maintained due to its computer-independent design. Besides, the MAF is easily installed, adapted if needed and the end-user does not need any specific skills or knowledge of a particular software. Finally we use the fact that the TEX language allows to mix both data processing and typesetting issues, maybe in a more complex way than usual word-processors and this is essential for the automatic production of forms.

Acknowledgment. I wish to thank all the people who have contributed to the MAF and proposed some extensions and corrections. Moreover, E. Schost and T. Zell have suggested some corrections to this article. Finally, I was glad to have benefited from the informations given on the WEB sites — maintained by volunteers — of the *Guilde des doctorants* and *Opération Postes* while looking for a job.

References

[1] AMS coversheet <`http://wwww.ams.org/coversheet`>.

[2] Guilde des doctorants <`http://guilde.jeunes-chercheurs.org/`>.

[3] D. Carliste. `keyval.dtx`, part of the *graphic bundle.*

[4] Ministère de l'Éducation Nationale (France) <`http://wwww.education.gouv.fr`>.

[5] Opération Postes <`http://smai2.emath.fr/postes`>.

Author's address:
Antoine Lejay
Projet OMEGA
Institut National de Recherche en Informatique et Automatique (INRIA)
& Institut Élie Cartan de Nancy (IECN)
Campus scientifique
BP 239
54506 Vandœuvre-lès-Nancy CEDEX, France
<`Antoine.Lejay@iecn.u-nancy.fr`>.

ŞäferTEX: Source Code Esthetics for Automated Typesetters

Frank-Rene Schaefer
(private research)
<frank_r_schaefer@gmx.net>

February 26, 2005

Abstract

In 2003, the first attempts towards ŞäferTEX were made, targeting to create a text processing with the goal of optimizing its ease of use, i.e. the beauty of its code appearance. The macro based TEX system still lives in niches of typesetting experts, computer scientist, and engineers who are willing to learn the overhead required to use the system - for the sake of high quality typesetting. ŞäferTEX faced the challenge to provide an interface language that does only differ minimally from a normal human edited text, while the compiler itself extracts the commands it requires.

Despite to simple wrappers programs, ŞäferTEX is a real three phases compiler, consisting of a lexical analyzer, a parser, and a code generator. In the last year the system mainly underwent internal changes that allowed to maintain this structure, while allowing a rather unusually simple and transparent programming syntax. As a consequence, the system now reached a robustness, so that it can be used by a wider audience.

This presentation shall give the reader an overview over the system of ŞäferTEX and demonstrate its abilities with a example application. Also, the fundamental ideas which allowed to maintain the classical three phases compiler structure are introduced. With the start of this conference the system can be downloaded at `safertex.sourceforge.net`.

The TEX Wrapper Structure: a basic TEX document model implemented in iTEXMac

Jérôme Laurens

November 5, 2004

1 Introduction

This presentation primarily concerns the high level user interface of the TEX typesetting system. In general, people find it difficult to work with TEX due to the powerful syntax, numerous auxiliary files created or managed, and the user interface that has very little in common with standard word processors. Moreover, sharing TEX documents with colleagues is often delicate as soon as some non standard LATEX is involved or, more frequently, there are some significant differences in the computer configurations. The purpose of this article is to lay the foundation for the TEX Wrapper Structure, which aims to help the user solve this kind of problems.

We first explain what could be the desiderata for a TEX document object model, then we give a precise description of the TEX Wrapper Structure, discussing the various solutions and the final choice. Finally, the concrete implementation used by iTEXMac[1] demonstrates an example of user interface.

An appendix briefly presents the latest developments concerning PDF synchronization which is a MacOS X specific feature of great interest for the whole TEX community.

2 A TeX Document Model

2.1 *De facto* document model

A document model aims to describe the storage and use of a certain kind of data: a simple document model might be a linear text, which is an ordered list of 8 bit numbers following the ASCII rules and stored in one flat file. More complex document structures are used either to describe data contents, for example Adobe's Portable Document Format, or to store them, for example old

[1]iTEXMac, one of the open source TEX front-ends on MacOS X, was presented during EuroTEX 2003 and TUG2004. Further information at `http://itexmac.sourceforge.net`

MacOS operating sytemes use a hierearchical file system with resource forks to allow file to store structured data. Regarding these two points among others, TEX is very specific mainly because it does not pose *a priori* any document model, letting the end user use its own *de facto* model. The question is to identify what core structure should have a TEX document model, that should be shared by quite all documents including the ones already existing.

Actually, a *self contained TEX document* is a series of files gathering data as various as images, linear text, formatted text, macro packages (LATEX style), code libraries (libjpeg...), engines (TEX, MetaPost) and their calling options. Of course this makes really huge documents, such that common parts are naturally eliminated, hoping that they will be available everywhere and every time one will ever need them. This results in some kind of *weak TEX document model* which has proved to be efficient, except in some rare situations where the syntax was broken by some package update, and less rare ones where engine options have been forgotten... Far-sighted TEX users carefully keep the various log files coming from typesetting because of the versioning information they contain. It is extremely helpful when fixing update problems, but still relies on non negligible human expertise where one could reasonably expect full computer assistance. When a strategy is available to record version information, it will be added to the TEX Wrapper Structure.

Generally speaking, a TEX document is composed of different kind of graphical objects, from linear text to pictures, possibly splitted into different files. There is no real problem concerning the various graphical data formats but the same does not hold for TEX source files. Any TEX user knows that a source document is not correct as long as it has not successfully passed TEX digestive process. More experienced users are perfectly aware of the problems that can appear when using certain combinations of macro packages. All this makes the data part of a TEX Document Model very difficult to define *a priori* in a complete and explicit description. This design, being as open as possible, is a real advantage because it provides quite unlimited document types. But at the same time, it does not take into account the document preparation stage and does not provide any help to the user in his real life struggle for document elaboration.

2.2 The meta information

For that purpose, advanced TEX dedicated editors have designed their proper strategy to assist the user with extraneous information not really necessary but missing when absent: the meta information. For example syntax attributes highlighting (marking TEX tokens, comments and other stuff with special colors) is a clever use of the information actually available as is in a source file. This can be improved by some syntax checking, that could mark bad commands just like the spell checker marks the misspelled words. For this to work efficiently in a real time context, we must collect the macros defined in the context and cache the whole dictionary list to improve access. Similarly, parsing the document contents for sectioning commands provides the user with a map that improves the overall sight and the navigation inside the document. All this is more or less filtering or interpreting the existing information to make it more accessible. Moreover, editors are free to add their own information if they think it is relevant. We can see that in fact, real TEX users may need more information that actually available in a TEX document, and TEX does not care about this kind of meta information. The TEX Wrapper Structure will mainly consider this point.

2.3 The document storage

As we must preserve actual TEX documents in a backward compatibility issue, we are only concerned with the document storage, more precisely the location where the different files are stored. Some of them must be located in definite folders, according to the TEX environment (in general following the TEX Directory Structure rules), while the user is absolutely free to name others. For them, some weak naming rules could help in their organization, without limiting their use. For example, people generally gather their graphic files in folders named images, graphics, pictures or whatsoever but there is not yet a widely spread strategy to become part of a TEX document model. Moreover, we must admit the use of different naming strategies to best fit the numerous situations one can imagine, for example, a unique image directory is certainly not advisable when the document is expected to contain thousands of logically organized images. Finally, the only naming rules we can safely state concerns the meta information and will be addressed by the TEX Wrapper Structure.

3 The TEX Wrapper Structure

We define the core TEX Wrapper Structure gathering information useful to any editor or utility, then we detail the TEX project concept and we briefly describe the concrete implementation developed in iTEXMac. This is a weak TEX document model given through a series of compliance rules, only assuming an underlying hierarchical file system. We also assume that all the document files but the standard macro packages are collected in one enclosing directory, but *a priori* different documents can share the same directory.

3.1 The core TEX Wrapper Structure

The only purpose of the core structure is to separate the document data, which is necessary, from the meta information, which is supplemental. Actually, the meta information is stored either in the very TEX source file (for example the %& first line trick to code for the format, the first commented line for TEXEXEC, emacs local variables to code for the string encoding, AUCTEX local variables in the file trailer), or an external file (the `.aux` LATEX file, the `Auto/` directory where AUCTEX caches its style attributes, the TEXniCenter projects) and each tool defines its own strategy without really taking care of one another. It is not yet the point to define a unique and complete set of meta information, but we are concerned with the storage location of the meta information. For practical reasons, it appears that some information such like the string encoding and the language should live near the document they are referring to, but other information including the list of project files and the root document identifier should live in a shared data base. If we consider all the tools of the TEX typesetting system, the simplest solution from the user point of view, is to collect the whole meta information into one central dedicated location. That way, no more meaningful comment will pollute the TEX source thus preserving the meta information from hazardous manipulations and preventing an innocuous TEX comment to become suddenly active while a utility has silently put some implicit information in it.

Finally, I strongly recommend not to use TEX comments anymore for anything else that commenting, except when conforming to a publicly available and widely accepted syntax rule.

3.2 The TEX project paradigm

With emacs' AUCTEX mode and TEXniCenter we already mentioned, the old DirectTEX pro is another example of editor that stores meta information in an external file. We propose to collect this information in one dedicated directory. So, a TEX project is just a directory named *document*.`texp` ("`texp`" stands for TEX Project) where shared or private meta information should be stored. The possible interference with already existing TEX documents is quite void because the `texp` extension is not yet used, this ensures a full backward compatibility.

To define the mapping linking projects to files, the TEX project is expected to maintain a list, either explicit or implicit, of all the files it is meant to manage. But conversely, it is not strictly necessary for the TEX source files to know the project they belong to (as for AUCTEX) because this information can be retrieved easily if we impose that TEX projects only manage files at the same level of below themselves in the file system hierarchy. Then, given a file path, we just have to scan the file hierarchy up to the root for TEX projects and only keep the appropriate ones.

The contents of the TEX project directory *document*.`texp` is described in the sequel. The user is not expected to view nor edit this data, so the format primarily concerns the programmers. More precisely, it is a balance between a flat XML file and an atomic directory structure, both suitable for information hierarchically organized. The "/" character is used as path separator.

- *document*.`texp/Info.plist` is an XML property list for a general purpose meta information wrapped in an `info` dictionary described in table 1 and subsequent tables. This is optional.

  We make use of the XML property list data format storage as publicly available at

  `http://www.apple.com/DTDs/PropertyList-1.0.dtd`

Key	Class	Contents
`isa`	String	Required with value: `info`
`version`	Number	Not yet used but reserved
`files`	Dictionary	The paths of the files involved in the project wrapped in a `files` dictionary described in table 2. It is an indirection table suitable for file name management. Optional.
`properties`	Dictionary	Attributes of the above files wrapped in a `properties` dictionary described in table 3, this is were string encoding and spelling key are recorded. Optional.
`main`	String	The *fileKey* of the main file, if relevant, where *fileKey* is one of the keys of the `files` dictionary. The main file is the one to be typeset or processed. Optional.

Table 1: `info` dictionary description where the TEX project maintains the list of known files, their properties and the main file identifier.

Key	Class	Contents
fileKey	String	The path of the file identified by the string *fileKey*, relative to the directory containing the TEX project. Each file key is unique. While the file name is subject to changes, the file key will never change: the latter is a strongly reliable file identifier. In general, no two different keys should correspond to the same path.

Table 2: `files` dictionary description: an indirection table particularly suitable for file name management.

It is indeed MacOS X centric but two PERL modules are available on CPAN to parse such XML files: Mac-PropertyList[2] andMac-PropertyListFilter[3]. Moreover, this can be changed in forthcoming versions without causing any harm from the user point of view.

- *document*.`texp/frontends` A directory dedicated to front-ends where they store private meta information.
- *document*.`texp/frontends/`*name* A private file or directory dedicated to

[2]`http://search.cpan.org/~bdfoy/Mac-PropertyList-0.9/`
[3]`http://search.cpan.org/~jgoff/Mac-PropertyListFilter-0.02/`

Key	Class	Contents
fileKey	Dictionary	Language, encoding, spelling information and other attributes wrapped in an `attributes` dictionary described in table 4. *fileKey* is one of the keys of the `files` dictionary.

Table 3: `properties` dictionary description: to each key identifying a file is associated a dictionary of attributes.

Key	Class	Contents
`isa`	String	Required with value: `attributes`
`version`	Number	Not yet used but reserved
`language`	String	According to latest ISO 639. Optional.
`codeset`	String	According to ISO 3166 and the IANA Assigned Character Set Names. If absent the standard C++ locale library module is used to retrieve the `codeset` from the `language`. Optional.
`eol`	String	When non void and consistent, the string used as end of line marker. Optional.
`spelling`	String	One of the *spellingKeys* meaning that the property list at *document*`.texp/`*spellingKeys*`.spelling` contains the list of known words of the present file wrapped in a spelling dictionary described in table 5. Optional.

Table 4: `attributes` dictionary description

Key	Class	Contents
`isa`	String	Required with value: `spelling`
`version`	Number	Not yet used but reserved
`words`	Array	The array of known words

Table 5: `spelling` dictionary description for the list of known words.

the front-end identified by *name*. The further contents definition is left under the front-end responsibility. The directory at

document`.texp/frontends/iTeXMac`

is reserved for iTEXMac private use, maybe AUCTEX can move its `Auto/`

directory into

document.`texp/frontends/AucTeX`

and TEXniCenter can use

document.`texp/frontends/TeXniCenter`.

This cooperative design is based on a strong separation of private meta informations from each other front-end, it prevents corruption and allows better recovery in case of error. Moreover, synchronization problems that may appear when two different utilities access the same flat file do not occur.

- *document.*`texp/users` is a directory dedicated to users and should not contain any front-end specific data. This is optional and reserved for further user.
- *document.*`texp/users/`*name* is a directory dedicated to the user identified by *name* (not its login name). Not yet defined, but private and preferably crypted.
- *document.*`texp/`*spellingKey.*`spelling` is an XML property list for lists of known words wrapped in a `spelling` dictionary defined in table 5 and uniquely identified by *spellingKey*. This format is stronger than a simple comma separated list of words. This is optional.

  We assume that a text document is multilingual and can have different spelling contexts, all of them being defined by a language with a dictionary and a list of known words. At this time, MacOS X programming interface does not allow to have more than one spelling context per open file, and the same might hold for other operating systems. So, each file is expected to have only one spelling context defined by a language and a spelling key, both defined in the `properties` dictionary (see the description in table 3). Then, a multilingual document will be splitted into files according to the language and the list of known words.

  Notice that there is no pre definite correlation between a language and a list of known words. And this design is certainly not the best we can elaborate, but it appears to be sufficiently efficient.

3.3 The TEX Wrapper Structure implemented in iTEXMac

The graphical user interface developed in iTEXMac takes benefit of the TEX Wrapper Structure. Private informations are cached to improve the user experience: window size and positions recording are the classical examples. Also, meta information about the engine and options used to typeset the document are stored, they are used to launch the appropriate utility with appropriate arguments assuming a teTEX like distribution is available. This should be shared once the latest TEX live is well established.

Technically, iTEXMac uses a set of private, built-in shell scripts to typeset documents. If this is not suitable, customized ones are used instead, possibly on a per document basis, but no warning is given then. No security problem has been reported yet, most certainly because such documents are not shared.

Notice that iTEXMac declares both `texp` and `texd` as document wrapper extensions to MacOS X, which means that *document*`.texp` and *document*`.texd` folders are seen by other applications just like other single file documents, their contents being hidden at first glance. Using another file extension for the TEX document will prevent this MacOS X feature without losing the benefit of the TEX Wrapper Structure and its TEX project.

4 Appendix: The `pdfsync` Feature

During the document preparation using the TEX typesetting system, the correspondence between the output and the original description code in the input is of frequent use, unfortunately it is not straightforward. Some commercial TEX frontends (Visual TEX[4] and TEXtures[5]) introduced a workaround. Then LATEX users could access the same features with a less-efficient implementation through the use of `srcltx.sty`, which added source specials in the DVI file. The command line option `-src-specials` now delegates that task to the TEX typesetting engine.

iTEXMac fully supports this synchronization allowing to jump from the DVI file to the `.tex` source and back. Moreover, Piero d'Ancona and the author have extended this feature from the `.tex` to the `.pdf` output. While typesetting a *document*`.tex` file with LATEX for example, the `pdfsync` package writes extra geometry information in an auxiliary file named *document*`.pdfsync`, subsequently used by the front ends to link line numbers in source documents with locations in pages of output PDF documents. iTEXMac, TEXShop[6] and TEXniscope[7] both support `pdfsync`.

The official `pdfsync` web site where file specifications and more complete explanations will be found at:

`http://iTeXMac.sourceforge.net/pdfsync.html`

Unfortunately, the various `pdfsync` files for Plain, LATEX or ConTEXt are not completely safe. Some compatibility problems with existing macro packages may occur. Moreover, sometimes `pdfsync` actually influences the final layout; in a case like that, it should only be used in the document preparation stage.

Notice that the `pdfsync` approach is different from Heiko Oberdiek's `vpe.sty`.

[4]`http://www.micropress-inc.com/`
[5]`http://www.bluesky.com/`
[6]`http://www.uoregon.edu/~koch/texshop`
[7]`http://docenti.ing.unipi.it/~d9615/homepage/mac.html`

Case Study of TEX in Commercial Data Based Publishing: Completely Automatic Typesetting of a Large Product Catalogue

Stephan Lehmke
QuinScape GmbH, Thomasstraße 1, 44135 Dortmund, Germany
Stephan.Lehmke@QuinScape.de
`http://www.QuinScape.de`

February 25, 2005

Abstract

In the talk, a *data based publishing system* for large product catalogues using `pdflatex` is presented which was developed to provide

1. top-quality typography;
2. completely automated document generation;
3. high flexibility for specifying data-based, design-oriented layouts;
4. multi-language support;
5. efficient production of very high volumes (number of documents, number of pages).

While some of the features (typographic excellence, multi-language support, support for high volumes) are provided by `pdflatex` 'out of the box' and at most require appropriate tweaking of TEX's parameters, to provide the optimal combination of completely automated document generation and high flexibility for document design and specification, a dedicated system consisting of several macro packages and document classes was created.

The process of designing and producing a 650 page product catalogue typeset in 14 languages is described in detail.

The **bigfoot** bundle for critical editions[*]

David Kastrup[†]

February 27, 2005

Abstract

The LATEX package bigfoot and supporting packages solve many of today's problems occurring in the contexts of single and multiple blocks of footnotes, and more. The main application is with philological works and publications, but simpler problems can be solved painlessly as well without exercising all of the package's complexities. For other problems not yet tackled in this area, a solid framework is provided.

1 Introduction

Footnotes in TEX are a problematic area. One reason is that TEX's insertion mechanism is far too basic to cope with more complicated usage patterns. Insertions are not subjected to the usual optimization methods of TEX, but instead are fitted on the page with a greedy algorithm at the time they are encountered. At that time, they may also be split or floated to the next page. A split does not take into account any mandatory following material on the vertical list: infinite values of `\widowpenalty` coupled with footnotes anchored in the next to last line will not be split at the correct point, and thus will have to get moved over to the next page.

Another deficiency is that when splitting a footnote, shrinkability is considered by TEX while doing the split, fitting more material on the page. However, at the time of the page break decision, the information about the shrinkability used for the insertion split gets lost, and consequently the page can appear overfull.

Since TEX does not even get the cases right for which it was designed, more complicated footnote schemes like those for critical editions have to be implemented mostly manually.

The `bigfoot` addresses a number of deficiencies and replaces the normal footnote mechanism.

2 Features

So what are the features that `bigfoot` provides?

- Multiple footnote apparatus[1] are possible.[2]
- Footnotes can be nested.[3]
- Footnotes are numbered in the order they appear on the page, and numbering may start from 1[†] on each page. In each apparatus, the footnotes are arranged in numerical order identical to page order. This does not sound exciting at all until you consider the implications of footnotes being nested: if the main text has some footnote[4] and then the publisher comments the main text with a footnote,[a] the logical order of footnotes (in which they appear in the source text) would have been to let footnote e appear before footnote a. The footnotes instead will be reordered to page order.[5]
- Footnotes may contain `\verbatim` commands[6] and similar, and they will just work as expected. This is achieved in a manner similar to the `\footnote` command of plain TEX.
- Footnotes can be broken across pages.[7]

[*]and a lot of other footnote applications

[†]`dak@gnu.org`

[1] An apparatus is one block of contiguous footnotes forming a logical and physical unit. Separate apparatus[b] can be independently broken to the next page.

[2] Actually, `manyfoot` already provides this functionality[c] but it fails to address a number of intricacies inherent to this sort of setup, a few of which follow.

[3] You can anchor footnotes for some apparatus in the main text[d].

[†] or whatever the first footnote symbol happened to be

[4] such as shown in this example footnote[e]

[5] The style file `perpage` has been extended with additional functionality for reordering such numbers.

[6] even stuff like `\verb-\iffalse-`

[7] While this does not sound like something excitingly new, it must be noted that TEX does not do a satisfactory job at splitting insertions, the underlying mechanism for split footnotes. In particular, TEX only manages to find a split when no material whatsoever is added to the page after the occurence of the split footnote. This might include another footnote in a different apparatus, or simply a line tied to the current line with an infinite penalty, for example because of a respective setting of `\widowpenalty`. In contrast, `bigfoot` breaks footnotes properly in such circumstances, and it uses a backtracking algorithm (with

[a] This is a subsequent comment to the main text. [b] Yes, this is the correct plural form. [c] and is loaded by `bigfoot`

[d] or any apparatus preceding it on the page

[e] which happens to have a comment attached to it. Notice that `bigfoot` will prefer to leave this smaller footnote block intact, as breaking it will not help fitting the above footnote block on the page.

- When footnotes are broken across pages, the color stack is maintained properly. Color is handled in LATEX with the help of specials that switch the color (and, in the case of `dvips`, restoring it afterwards with the help of a color stack). Restarting the footnote on the next page with the proper color is something that has never worked in LATEX. Now it simply does.

- Footnotes may be set in a compact form in one running paragraph.[8]

- Split footnotes will not get jumbled in the presence of floats. `bigfoot` is not afflicted by this bug in LATEX's output routine since it does not delegate the task of splitting footnotes to TEX in the first place. While the faulty output routine of LATEX may still jumble the order of footnotes in that particular case (when one footnote gets held over as an insertion 'floated' at infinite cost), `bigfoot` will sort the jumbled footnotes back into order before processing them.

- Each footnote apparatus can have its own private variant of `\@makefntext` and a few other macros and parameters responsible for formatting a footnote block. The default is to use what the class provides, but special versions can be defined, for example,

```
\FootnoteSpecific{variants}%
\long\def\@makefntext#1{...
```

  for the footnote block called "variants".

3 Drawbacks

What about current drawbacks?

- ε-TEX is used throughout. After it became clear that the implementation of the package would not be possible without using some of ε-TEX's features, its features were extensively employed: rewriting the package to get along without ε-TEX would be very hard, even if you came up with ideas for those cases where I could find no other solution. Free TEX distributions have come with ε-TEX for a long time by now (in fact, ε-TEX is now the recommended engine for LATEX, and actually used as the default in the latest TEX Live), but proprietary variants may lack ε-TEX support. The same holds for quite a few $\mathbf{\Omega}$ versions.

- The licence is not the LPPL, but the GPL. In my book, I consider this an advantage: the functionality of the package is quite important, and it is in its infancy yet. I would not like to encourage a market of proprietary offsprings directly competing with it. While with sufficient financial incentive I might feel confident enough to have the means to reimplement whatever noteworthy extension somebody else might come up with, at the current time I prefer this way of ensuring that the free development does not fall behind and that there is no incentive to turn to developers with no qualms about creating proprietary versions.

- `bigfoot` requires twice as many box registers[9] as `manyfoot`: one set in the form of an insertion for each footnote apparatus, one set as mere boxes.

- It can't handle more footnotes in a single block per page than the group nesting limit of TEX, and

early pruning of branches that can't beat the current optimum) for finding the best split positions for several footnote apparatus in parallel. The fill level of the page is taken into account as well as the costs of the individual splits. A split footnote is penalized with a penalty of 10000 (which is pretty similar to what TEX itself does when dealing with footnotes), so that in general TEX will tend to avoid splitting more than a single footnote whenever possible. One complication is that if the parts broken to the next page contain footnotes themselves, those have to be moved to the next page completely and adapted to the numbering of footnotes there[a]. This rather intricate and complicated mechanism leads to results that look simple and natural.

[8] While `manyfoot` and `fnpara` also offer this arrangement, `bigfoot` offers a superior solution in several respects:

- The line breaking can be chosen much more flexibly: with appropriate customization, it is possible to fine-tune quite well when and where stuff will be placed in the same line, and when starting a new line will be preferred.
- In-paragraph footnotes can be broken across pages automatically, just like normal footnotes. They will only be broken after the last footnote in the block has started.
- Pages will not become over- or underfull because of misestimating of the size of in-paragraph footnotes. Also the total width of such footnotes is not restricted to `\maxdimen` (which sounds generous at something like 6 m or 19 ft, until you realize that a few pages of text suffice to burst that limit, and a few pages of text are reached easily with longer variants of the main text). While TEX will accumulate boxes exceeding this size without problem, it panics at its own audacity if you actually ask about the total width of the acquired material. While one may still not have material exceeding a total *vertical* size of `\maxdimen` accumulate in one footnote block, one would usually need a few dozen pages for that, and so *this* limitation is much less noisome than the corresponding restriction on the horizontal size.
- The decision of whether to make a footnote in-paragraph or standalone can be changed for each footnote apparatus at any time, including on mid-page. In fact, you can make this decision for each footnote separately. Since display math requires vertical mode footnotes, this is convenient.
- `bigfoot` will make a good-faith effort to adapt the normal footnote layout provided by the document class with the `\@makefnmark` and `\@makefntext` macros to in-paragraph footnotes.

[a] which can be completely different!

[9] Since ε-TEX has an ample supply of box registers (32767 instead of 256), this is not really much of an additional limitation.

that is usually hardwired at 255.[†]

- Since it meddles considerably with the output routine's workings, interoperation with other packages doing the same might be problematic. Considerable effort has been spent on minimizing possibly bad interactions, but the results might not always be satisfactory and, at the very least, might depend on the load order of packages.
- It slows things down. This is not much of a concern, and usually the package is astonishingly fast.
- The complexity of the package makes it more likely for things to go wrong in new ways.[10]

4 Additional new packages

The bundle provides some more packages: `perpage` is used for the sort of renumbering games mentioned before, and `suffix` is used for defining augmented commands.

As an example of use for those packages we had previously a few examples where numbers like 7[‡] and 255[§] were given footnotes, and in order not to confuse this with powers as the following 666[11] is in danger of, we have switched to per-page numbering of footnotes with symbols for that purpose. The source code simply uses

```
like~7\footnote'{a lucky number}
```

namely a variant footnote command. How is that achieved? Just with

```
\newcounter{footalt}
\def\thefootalt{\fnsymbol{footalt}}
\MakeSortedPerPage[2]{footalt}
\WithSuffix\def\footnotedefault'{%
  \refstepcounter{footalt}%
  \Footnote{\thefootalt}}
```

A new counter is created, its printed representation is set to footnote symbols, the counter is made to start from 2 on each page (since symbol 1[¶] is a bit ugly), and then a variant of `\footnotedefault` is defined which will step the given counter and use it as a footnote mark.[12]

That's all. One can define several suffixes, the resulting commands are robust[13], and one can use arguments and other stuff. For example,

```
\WithSuffix\long\def\footnotedefault
   [#1]{#2}{...
```

would augment the macro `\footnotedefault` by a variant accepting an optional argument.

5 Some internals

5.1 Basic operation

The package uses most of the interfaces of `manyfoot` for its operation. While it uses TEX's insertions for managing the page content, the material collected in those insertions is in a pretty raw state and its size is always overestimated.[14] The actual material that goes onto the finished page is generated from the insertions at `\output` time.

Material that is put into insertions is prewrapped into boxes without intervening glue.[15] The box dimensions are also somewhat special: while the total height (height+depth) corresponds to the actual size of the footnote, the depth contains a unique id that identifies the last footnote in each box (of which there usually is just one, unless we are dealing with the remnants of an in-paragraph footnote apparatus broken across pages). The width is set to a sort key that is used for rearranging the various footnotes into an order corresponding to their order of appearance on the page.

The boxes are sorted by unvboxing them and then calling the comparatively simple sorting routine (a straight insertion sort):

```
\def\FN@sortlist{{%
  \setbox\z@\lastbox
  \ifvoid\z@ \else
    \FN@sortlist \FN@sortlistii
  \fi}}

\def\FN@sortlistii{%
  \setbox\tw@\lastbox
  \ifvoid\tw@\else
    \ifdim\wd\tw@<\wd\z@
      {\FN@sortlistii}%
    \fi
    \nointerlineskip \box\tw@
  \fi
  \nointerlineskip \box\z@}
```

If you find yourself running out of insertions, `etex` offers the `\reserveinserts` command.

[†] This limit seems sufficient at first glance, but one could use the various mechanisms available in connection with in-paragraph footnotes to make sure that a footnote will be broken across the page at a point closely related to the main text's breakpoint (for example, if you are doing an interlinear translation in a footnote). In that case, this limit might become problematic.

[10] Most of those problems should arise under requirements that could not possibly be met without the package, so this would be reason for improving rather than not using the package.

[‡] a lucky number [§] well, almost as lucky

[11] strange, yes? [¶] which is $*$

[12] `manyfoot` defines a two-argument command `\Footnote` that takes a footnote mark and corresponding footnote text.

[13] as long as their suffixes are so as well

[14] `bigfoot` simply sets each footnote, even those that should be typeset with others in one block, separately in its own paragraph for estimating its size, which should be a safe upper limit for the size a footnote can take when set in a paragraph with others.

[15] That way, there is never a legal breakpoint in an insertion.

and then all consecutive runs of hboxes are joined into vboxes. The desirability of breaking between two in-paragraph footnotes depends on their respective size, on whether this would save lines when typesetting, on whether a footnote apparatus can be shrunk by more than a certain factor in this manner, and whether the ratio of allowable joints between footnotes[16] to the number of footnotes exceeds a certain ratio.[17] The criteria are configurable per apparatus or globally.

There are some footnotes where a vertical arrangement is mandatory,[18] and the footnote must not be set into a hbox to start with. This is the case, for example, for footnotes containing display math. Placing a `+` sign before the opening brace of the footnote text will achieve that, and similarly a `-` sign can be used for switching in an otherwise vertically arranged footnote apparatus to horizontal arrangement.

`bigfoot` hooks into the output routine and does its accounting work before the main output routine gets a chance to get called. This work involves sorting the various contributions to a single insertion, joining together all in-paragraph footnotes into a single paragraph, measuring the resulting boxes, and gathering more material from the page in case that this produces an underfull box. Since the insertions `bigfoot` uses are unsplittable, this will often lead to an overfull box. In that case, the various footnote blocks get split to an optimum size before the real output routine gets called, and if this results in an underfull box again, more material gets called in again.

5.2 Dissecting `\@makefntext`

The footnote layout of document classes is given by `\@makefntext`. This macro receives one argument, the body of the footnote. We'll now discuss several problems we want to tackle in the context of using `\@makefntext` for implementing the layout prescribed by the class file.

5.2.1 Robust footnotes

One problem with LaTeX's footnotes is that they scan their arguments prematurely. We want them to behave more like those of plain TeX, to forestall complaints when `\verb` and its catcode mongering cousins fail to work in footnotes. The trick is to have the macro argument of the `\footnote` macro not really be a macro argument, but the content of an `\hbox` or `\vbox` command, and have subsequent code do its work with `\aftergroup`, once the command finishes.

This means that we have to cut `\@makefntext` into parts before and after its argument. It turns out that cutting the part before it starts processing its argument is rather easy:

```
\@makefntext \iffalse \fi
```

will do that. It executes and expands `\@makefntext` until it comes to the point where it would process its argument, which happens to be `\iffalse`, and then kills the rest of `\@makefntext`. At least as long as the argument `#1` does not happen to be in itself inside of a conditional, in which case bad things will happen. Very bad things. But a pretty thorough sampling of `\@makefntext` variants on TeX Live did not turn up such code.

Much more problematic is getting hold of the second part of `\@makefntext`. It turns out that about 95% of the variations out there in different class files will work with

```
\expandafter \iffalse \@makefntext \fi
```

which looks rather similar to the above. Unfortunately, it is not quite equivalent, since in the upper code, `\@makefntext` is cut into two once it has been expanded up to its macro parameter, whereas in the lower version it is cut into two before any parts of it get expanded. If any of the closing braces that follow `#1` in the definition of `\@makefntext` happen to belong to the argument of a macro starting before `#1`, they will cause spurious closing groups.

Getting the closing part at the end of the footnote without any remaining macro braces is more tricky, inefficient and error prone. One possibility is starting another instance of `\@makefntext` inside of a box to be discarded later. Then as its macro argument you use code that will repeatedly be closing opened groups until the outer group level is reached again and the box can be discarded. ε-TeX's grouping status macros (`\currentgrouplevel` and `\currentgrouptype`) make it possible to know how to close the current group and whether it is the last involved one. After everything that has been opened has been discarded again, the remaining tokens in the input stream should form a perfect complement to the tokens that the initial `\iffalse` trick has discarded at the start of the footnote.

One other mechanism probably worth playing with is the use of alignment templates, since they provide a natural way of having TeX switch input contexts across groups. The best approach in that regard would seem to parse the content of the footnote within a `\noalign` group of a `\valign`, but that still suffers from the problem that no automatic discretionaries are generated for explicit hyphens.

[16] where both footnotes around the breakpoint are considered potentially horizontal material

[17] A footnote apparatus in which there are just few horizontally arranged footnotes would appear inconsistent.

[18] like footnotes containing

- list environments
- display math like

$$\sum_{n=1}^{\infty}\frac{(-1)^n}{n} = \log\frac{1}{2}$$

But since most of the the `\@makefntext` variants out in the field are covered with the simple variant (basically, this is the case for all definitions that do not use `#1` within a macro argument itself), `bigfoot` for now has not added any of the more complicated versions. The group discarding trick might perhaps be made available with a separate package option at a later time, if there is sufficient demand for it.

But it may be easier in most cases simply to rewrite the culprits: after all, `\@makefntext` is rarely complicated. Most notably, the `\@makefntext` of the `ltugboat` class is so ridiculously contorted that the automated analysis of it fails. (It has been replaced with an equivalent for this article.)

5.2.2 `\@makefntext` in 'para' footnotes

is really a bit out of place: the 'para' footnote style sets all footnotes within one continuous running paragraph, a manner of operation quite different from the original intent of `\@makefntext`. Single footnotes are first collected in horizontal mode, and at `\output` time the relevant footnotes making it to the current page are pasted together. This has several problems: for one, `\@makefntext` will set paragraph breaking parameters. We need these at the time that we assemble the footnotes into one paragraph. But `\@makefntext` also generates the footnote mark, so we need to call it for each footnote.

So even when we set `\@thefnmark`[19] equal to an empty string at footnote assembly time, the assembled footnote mark will likely take up some additional space. This is not the end of our worries: while the formatting will be right for standard footnotes, it does not cater for 'para' footnotes. If we want to have a reasonably looking turnout, here are the conditions we have to meet:

1. At the beginning of the footnote block, or if a footnote starts right after a line break, the specified formatting should be used.

2. Within the line, we shall keep the spacing between footnote mark and footnote text correct. However, most styles right-justify the footnote mark within a box of fixed size. If we keep this sort of formatting, we will end up with a large space before short footnote marks, and a small one before longer marks. Since the amount of whitespace inside of a line should not be so large as to cause unsightly white holes, nor so small to make the footnote mark confused to be a part of the preceding footnote, we want a fixed spacing before the footnote mark.

The solution to these problems is to do a few measurements: we measure the width that an empty footnote mark would cause in the footnote box (and start our assembled footnotes with a negative space compensating that), and we typeset the footnote mark once on its own with `\@makefntext`, fishing with `\unskip` and `\lastbox` for the footnote mark box and resetting it to its natural size (which will kill the particular justification prevalent in the majority of class files doing justification). The difference in box size gets recorded separately until the time that the footnote gets set, and then the interfootnote glue is calculated accordingly.[20]

5.2.3 Maintaining the color stack

is not nice.[21]

What is the color stack, anyway? LaTeX's `color` package provides color selection commands that will change the current text color until the end of the group, where it will be restored.

The involved macros are

`\color@begin@group` is called at the start of each 'movable' box: material that does not necessarily appear right away. Without color support loaded, this does nothing. With color support loaded, it is usually equal to `\begingroup`.

`\color@end@group` is the corresponding macro at the end of 'movable' boxen. Any color restoration initiated with `\aftergroup` in the box will happen right here, still within the scope of the box, instead of outside where it would not move with the box.

`\set@color` will be called for setting the current color. It will also use `\aftergroup` in order to insert a call to `\reset@color` when the group ends.

`\reset@color` will restore the current color to what it was before the current group.

How will the color be restored? We have two different models:

dvips restores colors by making use of a color stack: `dvips` can 'push' a new color onto the stack, and pop the previous color back. Consequently, `\reset@color` inserts a special that tells `dvips` to pop the stack once.

pdftex instead restores colors by reinstating the color stored in `\current@color` after closing the group.[22]

It is clear that the `pdftex` model is insufficient to even keep the color of the main text across page breaks,

[19] the mark as displayed in the footnote

[20] A few classes work with `\parshape` or `\hangindent`, either directly or with a `list` environment, and this is also taken into consideration as far as possible.

[21] The main philosophy for work on the color stack has been summarized well by David Carlisle: "It's not my fault."

[22] Of course this means that if we are at the end of a movable

since on the next page there is no special after the page break that could switch back to the text color after the page footer[23] from the last page and headers from the current page have been placed with a default color.[24]

But in the context of footnotes, the problem is severely exacerbated: a footnote can be broken right in the middle of a sequence of color changes. The technically sound solution would be to switch to a different color stack for each footnote block. Since dvips does not offer multiple color stacks (and pdftex does not even offer a single one), we have to revert to trickery.

At each color change, the complete state of the color stack gets recorded in a mark. When the footnote is broken, we use the information in the mark in order to unwind the color stack to the state on the page before the footnote was entered. When the footnote is continued on the next page, the unwound color stack is reinstated again. Whenever `\color@begin@group` is called, the whole recording and restoration business is stopped (since a new context has been started), the record of the color stack essentially restored to empty, and only resumed when the corresponding group has ended.

In order to keep these proceedings fit for consumption by the general public, the reader is referred to the actual code for further details.

6 Outlook

At the time this article was written, quite a few tasks remained to be done. Further improvements in the footnote breaking decisions and their scoring metrics are needed. Flushing footnotes out in the middle of the page for short successive works would be nice. Amending footnotes with marginals (including line numbers) in a manner consistent with the main text would seem desirable. Additional footnote arrangements apart from the existing basic two styles should be easily implementable on top of the general scoring and breaking mechanisms.

7 Conclusion

It is hoped and expected that this bundle will become a basic building block for critical typesetting applications. While there are other packages available for that purpose, `bigfoot` (with its companions) offers the following important features:

- It is completely layout-neutral: while most solutions for critical typesetting are provided in the form of document classes, `bigfoot` does not make layout decisions but instead just uses the layout provided by a base class.
- Footnote arrangement and balancing is vastly superior to and more flexible than any of the available solutions.
- Color works.
- The interfaces for creating new functionality focused around footnotes are reasonably simple.

At the time this article was written, not all interfaces have yet been cast into stone. However, `bigfoot` can be mostly used as an upwards-compatible drop-in replacement of `manyfoot`.

One can define a `plain` footnote style in the manner of `manyfoot`, and then the default footnotes will get replaced by this footnote style. In fact, if one does not redefine the `plain` style, `bigfoot` will do so itself. Thus just loading it without any further action on behalf of the user will cater for the most common problems in connection with footnotes.

At the current point of time, still problems remain to be tackled: the accounting of page space and page splits was modeled after TEX's insertion mechanism and suffers from the same problem with regard to shrinkability, so in this paper, shrinkability has been removed from footnotes, a bad temporary hack. Page breaks currently are calculated by looping inside of the output routine instead of restarting it. In consequence, the headlines are not correct when material gets pushed to the next page. In a similar vein, floats like tables and figures might appear too soon. This will get solved with the next iteration of the package, after which a regular release should be possible.

It is not entirely clear how to deal satisfactorily with floats: if the first page size calculation results in a float being moved to the next page, and then it is determined that enough space on the current page is available for placing the float, doing so will significantly *reduce* the available space for the main text.

References

[1] `http://sarovar.org/projects/bigfoot`
(developer site and CVS instructions)

[2] `CTAN:macros/latex/contrib/bigfoot`
(released packages)

box, the restored color will be that at the time the box was assembled, not at the time it was used.

[23] and footnotes

[24] Heiko Oberdiek's `pdfcolmk` package tries to deal with that particular problem.

No abstract available

Tutorial TEXPower — Dynamic Presentations with LATEX

Stephan Lehmke

QuinScape GmbH, Thomasstraße 1, 44135 Dortmund, Germany
Stephan.Lehmke@QuinScape.de
`http://www.QuinScape.de`

February 25, 2005

Abstract

The tutorial introduces the TEXPower bundle for LATEX, providing an environment for designing *dynamic* ***`pdf`*** *presentations*, mainly for the purpose of displaying with a video beamer.

The heart of the bundle is the `texpower` package, providing commands for *incremental display* of page contents, commands for designing page backgrounds and 'panels', and commands for *navigation helpers*.

A further important part of the bundle is the `tpslifonts` package, providing 'display-friendly' font settings.

As the effects provided by `texpower` are implemented entirely based on the LATEX kernel, without resorting to special effects like PostScript, TEXPower is independent of the method of *pdf generation* and doesn't rely on external postprocessors or such. It is also completely independent of the *document class* used, though `seminar`-based classes harmonising well with the `texpower` package are part of the bundle.

The tutorial will cover, in varying depth, the following subjects:

1. Overview of the TEXPower bundle
2. The `powersem` class
3. The `tpslifonts` package
4. `texpower` General Features
5. `texpower`'s Color Handling
6. Page backgrounds, Panels
7. Navigation helpers
8. Incremental display
9. Designing a Presentation
10. Typical Applications

ε_{χ}TEX – Under the Hood

Gerd Neugebauer Michael Niedermair

February 26, 2005

The ε_{χ}TEX project aims at providing a reimplementation of TEX which is really open for extensions. Right from the beginning some design limitations of TEX have been dropped.

One base feature of ε_{χ}TEX is its configurability. It is possible to add new primitives by providing the implementation and register them in the configuration – no recompilation is required. This can even be performed from within the TEX macro language, if this feature is enabled.

We will show how the internals of ε_{χ}TEX work and which possibilities exist to extend ε_{χ}TEX and experiment with new features.

No abstract available

TEXLive2004 Windows Installer

Staszek Wawrykiewicz

March 1, 2005

Abstract

As I have learnt from different mailing lists, there are still so many users sharing not only the sentiment to fpTEX, but also prefer having the TEX installation conforming to the news in TDS/teTEX/web2c world. Yeah, it seems that also so many Polish users still love fpTEX/TL for MSWindows...

I know that most of users received the TEXLive 2004 collection, but this time, unfortunately, the Windows installer (usually the same as for fpTEX) is missing for some reasons. Anyway, I'd like to admit that MSWindows port included in TEXLive2004 works very well with all that news introduced in TDS 1.1 and TEXLive (thanks to Fabrice Popineau). What could the users (and followers of fpTEX) do with TEXLive2004? I'd like to announce the *provisional* TEXLive2004 installation program for Windows users by the author Pawel Jackowski:
`ftp://ftp.gust.org.pl/pub/GUST/contrib/TL2004/tlpm1.0beta.zip`

This program is dedicated for rather bold people (using sometimes the keyboard, instead of only the mouse), anyway it allows to install from the TEXLive2004-CD any scheme, collection or package, make listing of any scheme/collection/package, check dependencies and even uninstall the individual set, etc.

As this program doesn't touch in any way users' configuration (environment variables, registry), it can be safely applied for "just try it" purposes. All (important) configuration steps can be found in the '`tlpm.winconf`' file (sorry for that, but MSWindows is enough unpredictable, so we have to gather only some hints).

Installing and using Emacs, AUCTEX, RefTEX, preview-latex

David Kastrup

February 27, 2005

Abstract

The author's preview-latex package for WYSIWYG editing of LATEX constructs is slated for inclusion into AUCTEX right now. AUCTEX is by far the most prominent editing mode for LATEX and other TEX formats for use with Emacs and XEmacs (and probably one of the most important document creation environments for TEX/LATEX/ConTEXt/Texinfo altogether on most platforms). It offers document-adapted command completion and insertion, automated analysis of style files, syntax highlighting and indentation for readable source code and excellent TEX shell capabilities for PDFTEX, Omega and other engines. New developments are a toolbar and several other goodies. Managing crossreferences and bibliographies is a breeze with RefTEX, and additional tools like the symbolic formula manipulation system "calc" integrate nicely as well.

In this workshop, we will tackle installing Emacs as well as AUCTEX and preview-latex, configuring it for serious work, doing basic document creation taking a look at the possibilities Emacs offers on several platforms. We will work with the current development version of GNU Emacs, that is going to be released as 22.1 some time this year after almost 4 years of development since the last major new release.

Taking a look at the sparkling new bits and bytes fresh from the presses is going to prove both educational as well as exciting.

No abstract available

No abstract available

List of Authors

Bold papercodes indicate primary authors

Participants List

— A —

Jacques **André**
IRISA/INRIA-Rennes
Jacques.Andre@irisa.fr
Campus de Bealieu
F-35042 Rennes cedex
France

— B —

Benjamin **Bayart**
bayartb@edgard.fdn.fr
10 rue du Croissant
F-75002 Paris
France

Kaveh **Bazargan**
River Valley Technologies
kaveh@river-valley.com
9 Browns Court, Kennford
Exeter, EX6 7XY
United Kingdom

Nelson H. F. **Beebe**
University of Utah Department of Mathematics, 110 LCB
beebe@math.utah.edu
155 S 1400 E RM 233
Salt Lake City, 84112-0090
USA

Gábor **Bella**
ENST Bretagne
gabor.bella@enst-bretagne.fr
CS 83818
F-29238 Brest
France

André **Bellaïche**
Université Paris 7
abellaic@math.jussieu.fr
2, rue Pierre et Marie Curie
F-75005 Paris
France

Javier **Bezos**
Typesetter and Consultant
jbezos@wanadoo.es
C. Aldeanueva de la Vera, 15, 7-F
E-28044 Madrid
Spain

Thierry **Bouche**
Cellule MathDoc
thierry.bouche@ujf-grenoble.fr
B.P. 74
F-38402 Saint Martin d'Hères Cedex
France

Johannes Laurens **Braams**
TEXniek
johannes@braams.cistron.nl
Kersengaarde 33
NL-2723 BP Zoetermeer
The Netherlands

Klaus **Braune**
Universität Karlsruhe – Rechenzentrum
Klaus.Braune@rz.uni-karlsruhe.de
Zirkel 2 / Postfach 6980
D-76128 Karlsruhe
Germany

Gyöngyi **Bujdosó**
Faculty of Computer Science,
University of Debrecen
bujdoso@inf.unideb.hu
P.O. Box 12
H-4010 Debrecen
Hungary

— C —

Matteo **Centonza**
Società Italiana di Fisica
matteo@sif.it
Via Saragozza 12
I-40123 Bologna
Italy

Marie-Louise **Chaix**
EDP Sciences
mlchaix@edpsciences.org
17 av. du Hoggar
F-91940 Les Ulis
France

Hervé **Choplin**
Université de Tours
UFR Sciences et Techniques
choplin@delphi.phys.univ-tours.fr
Parc de Grandmond
F-37200 Tours
France

Jean-Louis **Colot**
Université Libre de Bruxelles, cp 238
Calcul Symbolique sur Ordinateur
jlcolot@ulb.ac.be
Boulevard du Triomphe
B-1050 Bruxelles
Belgium

— D —

Andreas **Dafferner**
Heidelberger Akademie der Wissenschaften
andreas.dafferner@urz.uni-heidelberg.de
Karlstr. 4
D-69117 Heidelberg
Germany

Bernard **Déchanez**
b.dechanez@bluewin.ch
Route de Chany 90
CH-1564 Domdidier
Switzerland

Christine **Detig**
J. Schrod Net & Publication Consultance GmbH
christine@detig.org
Kranichweg 1
D-63322 Rödermark
Germany

Andre **Dierker**
QuinScape GmbH
Andre.Dierker@QuinScape.de
Thomasstr. 1
D-44135 Dortmund
Germany

Luzia **Dietsche**
dietsche@gmx.de
Kissinger Str. 59
D-70372 Stuttgart
Germany

Michael **Doob**
Department of Mathematics
University of Manitoba
mdoob@ccu.umanitoba.ca
342 Machray Hall
Winnipeg, R3T 2N2
Canada

Karin **Dornacher**
DANTE e.V.
office@dante.de
Postfach 101840
D-69008 Heidelberg
Germany

— E —

Martin **Etter**
martin.etter@gmx.de
Bergstraße 5
D-70806 Kornwestheim
Germany

Christoph **Eyrich**
eyrich@math.tu-berlin.de
Skalitzer Straße 74a
D-10997 Berlin
Germany

— F —

Robin **Fairbairns**
rf@cl.cam.ac.uk
30 Mill End Road
Cambridge, CB1 9JP
United Kingdom

Hong **Feng**
RON's Datacom Co., Ltd.
fred@mail.rons.net.cn
Suite 3-3, WuZhong Str. 200, Don District
430040 Wuhan
China

Thomas **Feuerstack**
FernUniversität in Hagen
Universitätsrechenzentrum
Thomas.Feuerstack@fernuni-hagen.de
Universitätstr. 21
D-58084 Hagen
Germany

Jonathan **Fine**
The Open University
j.fine@open.ac.uk
Walton Hall
Milton Keynes, MK7 6AA
United Kingdom

Robert **Fischer**
derfischer@gmx.net
Am Krümmelweg 8
D-54311 Trierweiler
Germany

Daniel **Flipo**
U.S.T.L.
daniel.flipo@univ-lille1.fr
Cité scientifique
F-59655 Villeneuve d'Ascq Cedex
France

David **Fuchs**
drfuchs@yahoo.com
1775 Newell Rd.
Palo Alto, California, 94303
USA

— G —

Ralf **Gärtner**
ralf.gaertner@t-systems.com
Ötztalerstr. 5b
D-81373 München
Germany

Falk **Gerwig**
flak2k@gmx.de
Im Schlehbusch 9
D-75397 Simmozheim
Germany

Michel **Goossens**
CERN
michel.goossens@cern.ch
Departement IT
CH-1211 Geneve 23
Switzerland

Steve **Grathwohl**
Duke University Press
grath@duke.edu
905 W Main Street Suite 18B
Durham, NC, 27701
USA

Holger **Grothe**
TU Darmstadt,
Fachbereich Mathematik
grothe@dalug.de
Kittlerstraße 38
D-64289 Darmstadt
Germany

Patrick **Gundlach**
patrick@gundla.ch
Universitätsstraße 71
D-44789 Bochum
Germany

Michael **Guravage**
Literate Solutions
guravage@literatesolutions.com
Mijndensedijk 11a
NL-3632NT Loenen aan de Vecht
The Netherlands

— H —

Hans **Hagen**
PRAGMA
Advanced Document Engineering
pragma@wxs.nl
Ridderstraat 27
NL-8061GH Hasselt
The Netherlands

Thê Thành **Hàn**
University of Education
in Ho Chi Minh City
hanthethanh@gmx.net
280 An Duong Vuong
Ho Chi Minh
Vietnam

Yannis **Haralambous**
ENST Bretagne
yannis.haralambous@enst-bretagne.fr
CS 83818
F-29238 Brest
France

Jim **Hefferon**
St Michael's College
ftpmaint@tug.ctan.org
Box 285
Colchester, VT, 05439
USA

Laure **Heïgéas**
France

Oliver **Heins**
olli@sopos.org
Auf dem Brinke 1
D-30453 Hannover
Germany

Hartmut **Henkel**
hartmut_henkel@gmx.de
In den Auwiesen 6
D-68723 Oftersheim
Germany

Taco **Hoekwater**
Elvenkind BV
taco@elvenkind.com
Spuiboulevard 269
NL-3311 GP Dordrecht
The Netherlands

Morten **Høgholm**
morten.hoegholm@latex-project.org
Persillehaven 40, 1215
DK-2730 Herlev
Denmark

Klaus **Höppner**
DANTE e.V.
klaus@dante.de
Postfach 101840
D-69008 Heidelberg
Germany

Karel **Horàk**
Mathematical Institute,
Academy of Sciences
horakk@math.cas.cz
Zitna 25
CZ-115 67 Praha 1
Czech Republik

Sophie **Hosotte**
EDP Sciences
hosotte@edpsciences.org
17 av. du Hoggar - BP 112 -
Courtaboeuf
F-91944 Les Ulis Cedex A
France

Jean-Michel **Hufflen**
LIFC - Université de Franche-Comté
hufflen@lifc.univ-fcomte.fr
16, route de Gray
F-25031 Besancon CEDEX
France

— J —

Bogusław **Jackowski**
BOP
b_jackowski@gust.org.pl
Bora-Komorowskiego 24
PL-80-377 Gdańsk
Poland

Jean-Paul **Jorda**
EDP Sciences
jorda@edpsciences.org
17 av du hoggar
F-91940 Les Ulis
France

— K —

David **Kastrup**
dak@gnu.org
Kriemhildstr. 15
D-44793 Bochum
Germany

Jörg **Knappen**
jknappen@web.de
Dieselstraße 13
D-66123 Saarbrücken
Germany

Donald E. **Knuth**
USA

Thomas **Koch**
Dante e.V.
thomas@dante.de
Tempelstraße 20
D-50679 Köln
Germany

Harald **König**
koenig@linux.de
Königsberger Str. 90
D-72336 Balingen
Germany

Reinhard **Kotucha**
reinhard.kotucha@web.de
Marschnerstr. 25
D-30167 Hannover
Germany

Johannes **Küster**
typoma
info@typoma.com
Karl-Stieler-Str. 4
D-83607 Holzkirchen
Germany

— L —

Klaus **Lagally**
Universität Stuttgart
lagallyk@acm.org
Universitätsstraße 38
D-70569 Stuttgart
Germany

Joachim **Lammarsch**
Rechenzentrum der
Universität Heidelberg
joachim.lammarsch@urz.uni-heidelberg.de
Im Neuenheimer Feld 293
D-69120 Heidelberg
Germany

Marion **Lammarsch**
Psychologisches Institut der
Universität Heidelberg
marion.lammarsch@psychologie.uni-heidelberg.de
Hauptstraße 47-51
D-69117 Heidelberg
Germany

Dag **Langmyhr**
University of Oslo
dag@ifi.uio.no
PO box 1080 Blindern
N-0316 Oslo
Norway

Maurice **Laugier**
GUTenberg
laugier.maurice@tele2.fr
La Haute Tourronde
F-05000 Gap
France

Jérôme **Laurens**
Département de Mathématiques,
Université de Bourgogne
jerome.laurens@u-bourgogne.fr
9, avenue Alain Savary
F-21034 Dijon cedex
France

Matthieu **Laverne**
Black Media
`insert@easynet.fr`
43, rue du Commerce
F-75015 Paris
France

Stephan **Lehmke**
QuinScape GmbH
`Stephan.Lehmke@QuinScape.de`
Thomasstraße 1
D-44135 Dortmund
Germany

Martin Wilhelm **Leidig**
`Martin.Leidig@onlinehome.de`
Centgrafenweg 9
D-69181 Leimen
Germany

Antoine **Lejay**
INRIA Lorraine
`Antoine.Lejay@iecn.u-nancy.fr`
BP 239
F-54506 Vandoeuvre-les-Nancy CEDEX
France

Knut **Lickert**
`eurotex2005.9.knut68@spamgourmet.org`
Obertorstr. 61
D-73728 Esslingen
Germany

Manfred **Lotz**
`manfred@dante.de`
Sindlinger Str. 8
D-60326 Frankfurt
Germany

Thomas **Lotze**
`thomas.lotze@latex-project.org`
Ziegelmühlenweg 3
D-07743 Jena
Germany

Jerzy **Ludwichowski**
Polish TEX Users Group - GUST
`Jerzy.Ludwichowski@uni.torun.pl`
Plac Rapackiego 1
PL-87-100 Toruń
Poland

— M —

Lars **Madsen**
Department of Mathematical Sciences
University of Aarhus Denmark
`daleif@imf.au.dk`
Tousvej 97, 2.TV
DK-8230 Aabyhoej
Denmark

Gisela **Mannigel**
`gm@tesionmail.de`
Auenweg 2A
D-82407 Wielenbach
Germany

Lionel **Marcouire**
Dauphine
`lionel@marcouire.com`
25 rue de la Libération
F-92500 Rueil-Malmaison
France

Wendy **McKay**
Control and Dynamical Systems
California Institute of Technology
`wgm@cds.caltech.edu`
M/C 107-81 (1200 E California Blvd)
Pasadena, 91125-8100
USA

Frank **Mittelbach**
LATEX3 Project
`frank.mittelbach@latex-project.org`
Zedernweg 62
D-55128 Mainz
Germany

Mikael **Möller**
TEX-Försäljning AB
`texab@faksimil.se`
Kampementsgatan 34
S-11538 Stockholm
Sweden

— N —

Gerd **Neugebauer**
`gene@gerd-neugebauer.de`
Im Lerchelsböhl 5
D-64521 Groß-Gerau
Germany

Richard WD **Nickalls**
Department of Anæsthesia,
City Hospital, Nottingham
`dicknickalls@compuserve.com`
Hucknall Road
Nottingham, NG5-1PB
United Kingdom

Michael **Niedermair**
`m.g.n@gmx.de`
Am Malerwinkel 16
D-85778 Haimhausen
Germany

Janusz **Nowacki**
FOTO-ALFA
`j.nowacki@gust.org.pl`
Al. 23 Stycznia 54D
PL-86-300 Grudziądz
Poland

— O —

Cezary **Obczyński**
Faculty of Mathematics, University Łódż
`czacza@math.uni.lodz.pl`
Banacha 22
PL-90-238 Łódż
Poland

Heiko **Oberdiek**
`oberdiek@uni-freiburg.de`
Kärntner Weg 3
D-79111 Freiburg
Germany

— P —

Gilles **Pérez-Lambert**
Université Paul-Valéry, Montpellier III
`Gilles.Perez@univ-montp3.fr`
Route de Mende
F-34199 Montpellier Cédex 5
France

Éric **Picheral**
`eric.picheral@free.fr`
16 rue du docteur Ferrand
F-35200 Rennes
France

Karel **Piska**
Institute of Physics,
Academy of Sciences
piska@fzu.cz
Na Slovance 2
CZ-18221 Prague
Czech Republik

Fabrice **Popineau**
Supélec
fabrice.popineau@supelec.fr
2 rue E. Belin
F-57070 Metz
France

Christophe **Pythoud**
ABCIS Sarl
pythoud@abcis.ch
22, rue du Pont
CH-1003 Lausanne
Switzerland

— R —

Sebastian **Rahtz**
Oxford University
sebastian.rahtz@gmail.com
13 Banbury Road
Oxford, OX2 7BG
United Kingdom

Bernd **Raichle**
DANTE e.V.
bernd.raichle@gmx.de
Kissinger Str. 59
D-70372 Stuttgart
Germany

Henri **Rasolofomasoandro**
he.rasolof@libertysurf.fr
54 Chemin d'Arvillard
F-74160 Archamps
France

Thomas **Ratajczak**
ratajczak@web.de
Wredestr. 10
D-97082 Würzburg
Germany

Arthur **Reutenauer**
École normale supérieure
Arthur.Reutenauer@ens.fr
45 rue d'Ulm
F-75005 Paris
France

Denis **Roegel**
LORIA – University of Nancy 2
roegel@loria.fr
LORIA – BP 239
F-54506 Vandoeuvre les Nancy cedex
France

Christian **Rossi**
rossi@in2p3.fr
168 cours Lafayette
F-69003 Lyon
France

Chris **Rowley**
Open University / LATEX3
c.a.rowley@open.ac.uk
1-11 Hawley Crescent
London, NW1 8NP
United Kingdom

Erich **Ruff**
Erich_Ruff@T-Online.de
Kruckenburgstr. 21
D-81375 München
Germany

— S —

Volker RW **Schaa**
DANTE e.V.
volker@dante.de
Landwehrstraße 33
D-64293 Darmstadt
Germany

Frank-René **Schäfer**
Germany

Walter **Schmidt**
w.a.schmidt@gmx.net
Nürnberger Straße 76
D-91052 Erlangen
Germany

Joachim **Schrod**
J. Schrod Net & Publication
Consultance GmbH
jschrod@acm.org
Kranichweg 1
D-63322 Rödermark
Germany

Martin **Schröder**
martin@oneiros.de
Crüsemannallee 3
D- Bremen
Germany

Torsten **Schütze**
Siemens AG, CT IC 3
torsten.schuetze@siemens.com
Otto-Hahn-Ring 6
D-81739 München
Germany

Peter **Seitz**
p.seitz@ks-ingenieurconsult.de
Kleinreuther Weg 53
D-90408 Nürnberg
Germany

Martin **Sievers**
Martin.Sievers@Schaephuysen.de
Im Treff 8
D-54296 Trier
Germany

Karel **Skoupý**
ETH Zürich
skoupy@inf.ethz.ch
Manegg Promenade 136
CH-8041 Zürich
Switzerland

Petr **Sojka**
Masaryk University in Brno
Faculty of Informatics
sojka@fi.muni.cz
Botanicka 68a
CZ-60200 Brno
Czech Republik

Tobias **Sterzl**
DANTE e.V.
tobias.sterzl@gmx.de
Häldenstraße 30
D-75236 Kämpfelbach
Germany

Thierry **Stoehr**
AFUL - Formats-Ouverts.org
stoehr@aful.org
34 rue de l'écluse
F-77000 Melun
France

Péter **Szabó**
BUTE Department of Computer Science and Information Theory
pts@inf.bme.hu
Mũegyetem rakpart 3–9
H-1111 Budapest
Hungary

Jolanta **Szelatyńska**
Nicolaus Copernicus University,
IT Centre
Jolanta.Szelatynska@uni.torun.pl
Plac Rapackiego 1
PL-87-100 Toruń
Poland

— T —

Adam **Twardoch**
adam@twardoch.com
Logenstr. 2/301
D-15230 Frankfurt (Oder)
Germany

— V —

Erik **Van Eynde**
LUDIT — Katholieke Universiteit Leuven
Erik.VanEynde@cc.kuleuven.ac.be
De Croyelaan 52a
B-3001 Heverlee
Belgium

Ulrik **Vieth**
ulrik.vieth@tesionmail.de
Vaihinger Straße 69
D-70567 Stuttgart
Germany

— W —

Zofia **Walczak**
Department of Mathematics,
University of Łódż
zofiawal@math.uni.lodz.pl
Banacha 22
PL-90-238 Łódż
Poland

Sebastian **Waschik**
sebastian.waschik@gmx.de
Kätnerweg 13f
D-22393 Hamburg
Germany

Stanisław **Wawrykiewicz**
GUST
staw@gust.org.pl
Broniewskiego 2/2
PL-81-837 Sopot
Poland

Olaf **Weber**
olaf@infovore.xs4all.nl
Boulevard Heuvelink 1-11
NL-6828KG Arnhem
The Netherlands

Gerben **Wierda**
Gerben.Wierda@rna.nl
The Netherlands

— Z —

Hermann **Zapf**
Seitersweg 35
D-64287 Darmstadt
Germany