

Is there a need for T_EX&Co courses?

Much self-study material is available, to start with the T_EXbook and the MFbook, next to excellent tutorials. Courses are nevertheless necessary, IMHO, because of the complexity. Moreover, I consider self-study not sufficient, even dangerous.

Apparently the public does not ask for courses.

Education material on T_EX Live DVD

Nowadays, we have AnyT_EX&Co on our PCs at home, such as the L^AT_EX-biased Integrated Development Environment, T_EXnicCenter, distributed on the T_EX Collection DVD. However, as plain T_EXie, I missed in the output dropdown window T_EX→pdf and ilks. It should not be so difficult to add that, tho plain T_EXies form a minority. Please do. More difficult, I presume, is to provide for MetaPost processing from within T_EX. Both are necessary for my proposal with respect to a MasterClass. At the conference I became aware of T_EXshop on the Mac OS X+ and Jonathan Kew's T_EXworks IDE in the spirit of T_EXshop under Linux and Windows XP.

The public domain T_EX program, distributed on the T_EX Collection DVD, comes along with a wealth of documentation, software and books, which allow for self-study of the open and public domain programs AnyT_EX&Co, while for advanced questions one may consult FAQs, or discussion lists with their archived earlier discussions. I consider this a tremendous development, beneficial, except for the lack of standards in T_EXing, and that it is not enough for a casual visitor of the internet, who likes to get started, despite the excellent active installation PDF documents in your language. However, self-study can be dangerous, but... in the absence of courses there is no choice. A standard in T_EXing in the free T_EX world is most likely not realistic, alas.

But...
we might try.

Pluriform macro writing emerged, inhibiting readability, as the most obvious characteristic. No programming paradigms, despite Knuth's example macro writing in the T_EXbook: macros which constitute plain.tex, and macros to mark up a letter, concert, or a book, and his gkppic line-diagram macros in plain T_EX, related to L^AT_EXs picture environment,² which were used for type-

setting his Concrete Mathematics. No path nor picture datastructures, no color and no filling of arbitrary closed curves. The manmac macros were created to typeset the T_EXbook and ilks, and likely his The Art of Computer Programming oeuvre. In 4AllT_EX, in the TeX directory GENERIC, macros are collected, such as the midnight suite by van der Goot next to macros by Eijkhout, ... No stress on paradigms. The T_EX Collection DVD contains a copy of the Comprehensive T_EX archive, the CTAN. Searching with keyword BLUe yielded no match.³

Education issues

Education turns around: language, awareness, insight, T_EXnique proper, courseware, and the personality of the teacher.

Language

A fundamental aspect in general education is language, it is the basis. Everybody speaks a language, can read and write, and may publish easily nowadays via the WWW. Language serves a lifetime! Language fluency is a prerequisite for participation in a culture, is fundamental in communication. Language lies at the heart of publications.

The T_EXbook spends several chapters on just typesetting text, deals with the variety of

- type faces
- accented characters
- ligatures
- hyphenation
- line and page breaking,
- structuring commands ...

T_EX is well-suited for typesetting all sort of languages, not just those based on Latin fonts. A fundamental assumption, next to the basic boxes and glue approach, is that a paragraph is the unit for hyphenation, not the keyboarded lines of input are taken as lines for the output. T_EX neglects one eol-symbol in a row, treats it as a space. In a revision of T_EX, was it 1989?, the \language command was introduced to facilitate for various, language specific hyphenation styles.

T_EX arose from Knuth's dream to typeset mathematics beautifully. I presume Knuth was very much surprised by the complexity of typesetting ordinary language, automatically and foolproof.

Typesetting novels by T_EX is a trifle. However, book production is culture biased, with variations in layout

and the used fonts. Is \TeX the only tool available for this task?

Definitely not. Novels are produced by word processors, with excellent spelling checkers, I presume. I have heard of Adobe's Indesign, no hands-on experience as yet, however. MS Word I use quite often for contributions to our local gardening bulletin. These gardeners have not even heard of Any \TeX .

It amazes me that we have no Babel \TeX as yet, where the commands and error messages can be instantiated for your language. For example in Russian you would then end with $\backslash\text{пока}$ instead of $\backslash\text{bye}$, and just keyboard Cyrillics.

It also surprises me that we don't have 2 communicating windows open simultaneously: one for the editor and the another for the typeset result, as next best to WYSIWYG. Bluesky's \TeX enjoyed on the fly compilation, called flash mode.

But ...

\TeX works has also the edit and the pdf window open and linked, as I learned at the conference. When an error occurs one is directly led to the line with the error in the source file in the edit window.

LUGs The most astonishing aspect of \TeX being around is that there have arisen so many Language-biased \TeX user groups. This demonstrates a relationship between \TeX and languages. It is misleading to think that \TeX has only to do with typesetting Math. LUGs have proven that

a subset of \TeX can be used extremely well to typeset ordinary text.

Malevich
Suprematism:
White cross on a
White background
Emulation \rightarrow

Maybe this fact should be brought to the attention of a casual user, and should not be burdened by all the other complex tasks \TeX can do, which might frighten an innocent user.

I have included a picture, and its emulation, of Malevich⁴ because he is the father of suprematism, which deletes the superfluous, which I associate with Minimal Markup.

NTG was founded twenty years ago PCs were emerging. We could access the computer centre from home by telephone through 1024baud modems. NTG started a fileserver and the digest TeX-nl . UNIX was taking off.

No networks nor WWW were in sight. The mainframes or midi's were accessed via terminals. I think that the meetings twice a year and the Minutes and ApPendiceS, MAPS, appearing in time, had a great impact on NTG.

I contacted budding LUGs in Germany, France, England and Scandinavia, and looked for cooperation. We cooperated in organizing Euro \TeX s. In my time as president, NTG funded the $\text{L}^{\text{A}}\text{TeX}2\epsilon$ project. Much later NTG took part in funding the Latin and Gyre font projects.

Jargon Mathematics from the language point of view is a jargon, with its own symbols, structures, meanings and definitions. In typesetting Math \TeX is superb, still unmatched?⁵

But...

Microsoft, with its Cambria project and the use of OpenType Math fonts, may have taken the lead.

Other jargons can be typeset equally well, but you have to write the equivalent of the intelligent math mode yourself. I have not heard of such an effort.⁶

Jargon books are more complicated than novels, especially with respect to tradition in typesetting the jargon, such as: contents, back matter (an index, tables of contents, pictures, tables, . . . , references) and cross-links. For typesetting Math one must be a mathematician, have enjoyed a Math education, in short one must know the jargon.

But...

That is not enough, at least with the mathematical training I enjoyed. No typesetting conventions of math was ever taught to me. No Mathematical writing, 1989 by Knuth as main author (who else?) existed. Happily, \TeX embodies it all: the subtle spacing around Math symbols, the awareness of operators and how to typeset them in context, the composition of symbols with limits for example, is all taken care of by \TeX , though \TeX is not perfect. In-line Math and displayed Math may look different. The choices made by \TeX , implemented in the Math mode, are wired in, but... parameterized. Math constructs can be marked up by using macros. The Math fonts are not just ordinary fonts, they come along with a suite of $\backslash\text{fontdim}$ parameters for placement of the glyphs in context by the intelligent Math mode. Using

just another, commercial, Math font with the richness of the T_EX math mode knowledge is not trivial. OpenType Math fonts come also with a suite of parameters, some the same as in T_EX some different, and some beyond T_EX's. Work on incorporating OpenType Math fonts for use in T_EX, is underway by Bogusław Jackowski et al., and about to be released.

The above observations delineate the AnyT_EX&Co users who make full use of the power of T_EX \leftrightarrow MetaPost: those who want to typeset Math (and to a lesser extent other jargons) beautifully, and ... be in complete control. Physics jargon has a lot in common with Math, and I think the Physics typesetting tradition is highly similar.

Awareness

To be aware of what is available in order to choose the right tool for the task at hand is all important and costs-effective.⁷

To create awareness is a main objective of education, next to acquiring T_EXnique proper and learning how to cope with change and the unknown.

Awareness of competing and assisting tools is mandatory. As a WYSIWYG competing tool we have MS Word, with the Cambria module for Math, which uses the OpenType Math fonts. I glanced into the Math possibilities of Word 2007, and I'm impressed by what MS has achieved, especially with respect to user friendliness. Below, I have included a snapshot.

$$\begin{aligned}
 & \iiint_{\square} \square \\
 & \sum_{\square} \square \sum_{\square} \square \sum_{\square} \square \sum_{\square} \square \sum_{\square} \square \\
 & (x + a)^n = \sum_{k=0}^n \binom{n}{k} x^k a^{n-k}
 \end{aligned}$$

No confusing details, no markup tags which one must remember, just templates to be filled in, or modified. The dotted squares in the template formulas can be filled in with the needed symbols in a WYSIWYG way. Easy, isn't

it? It is hard to convince users that T_EX is better, I guess. The Binonium of Newton, with its 'n over k' and 'limits with the summation symbol', looks correctly typeset to me. Is T_EX more flexible? Rhetorical question.

But...

MS (Math) looks easier. T_EX&Co must watch out, the law of diminishing returns seems to apply.

However, in T_EXnicCenter I found similar but less advanced templates. if you click on an icon the LaT_EX code, will be inserted in your script, which saves you typing, and which relieves you from remembering the commands. You still have to look at the inserted tags for where to fill in. In MS the fill-in place is marked by empty dotted squares. For example, for the \rightarrow icon T_EXnicCenter inserts the LaT_EX control sequences

```
\stackrel{\square}{\square}\rightarrow
```

Do we have IDEs with really advanced editors with AnyT_EX support, which not only prompt markup tags, but also prompt formula templates to be filled in?

With respect to WYSIWYG, we compromise by providing two communicating windows open: the editor window with the source file and the typeset window with the pdf result.

At the conference attention was paid to provide support for OpenType (Math) fonts for use in T_EX.

Lack of awareness shows up when how to do typesetting tasks are published over and over again, without taking notice of, or mentioning, earlier work nor giving proper credits. Is the T_EXworld anarchistic? In the last issue of MAPS, spring 2009, I read about how to typeset an addition table, which is similar to Pittman's approach of long ago of typesetting by T_EX a multiplication table. The author did not mention earlier work if not by me, while the title alluded to the past. It is true that it was typeset in ConT_EXt, and that is new, I guess. Superfluous in view of my plea to provide macros for creation of the table data in plain T_EX to be used in AnyT_EX, though in this example case the data are a trifle.

Ignoring the past is not a scientific tradition.

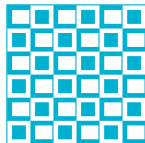
I have included below the essentials of my (plain, what else?) macros for typesetting a multiplication, addition, ... table, of a decade ago, as supplied in my Publishing with T_EX guide, PWT, which accompanies blue.tex. The invoke reads

```
\def\dataMT{1\cs 2\cs 3\rs
          2\cs 4\cs 6}
$$\framed\ruled %...Attributes
\bluetable\dataMT$$
```

The creation of the (general) data can be done as follows

```
% Creates 1 2 3
%          4 5 6
\def\rows{\c1
  \cols \advance\c1
  \ifnum\c1>\mr \swor\fi
  \rs\rows}
%
\def\cols{\te\r\multiply\te\c\the\te
  \advance\c1
  \ifnum\c1>\mc \sloc\fi
  \cs\cols}
%
\def\sloc#1\cols{\fi}%terminator
\def\swor#1\rows{\fi}%terminator
\def\rs{\par}%row separator
\def\cs{ } %column separator
%
\mr2 \mc3 \rows %invoke 23 table data
```

T_EX macro writing of the past, the present, or the future?



One might argue that it just generates the data, and that the complexity of markup is absent.

This is done on purpose adhering to the separation of concerns adage. More general, in my bordered table macros, I first generate the data and then do with the data whatever I have to do. A consequence of this approach is that the border of a table, which contains usually information about the entries, is handled separately. In blue.tex a table is composed of border, data, and caption or footer, much in the spirit of Knuth's \bordermatrix. By attributes one can specify framing and ilks.

My minimal markup macro for creation of the data for the multiplication table is like Knuth's robust macros timeless, and that is what I ask you to do: write timeless, robust, mean and lean macros in plain T_EX, the lowest common subset

of all T_EXs, to be reused in AnyT_EX.⁸ However, in this special case the data can just be supplied, but that is not the issue. OK, you are right we should start with creating a library of reusable parts.

Awareness of other tools Phil Taylor in his recent \parshape pre-processor⁹ starts with telling that he used HTML, because 'HTML can flow text around an embedded (rectangular) image in a totally straightforward manner, requiring nothing more than...'. He continues that he would like to use T_EX and provided macros, well... his pre-processor. T_EX has a more powerful, general mechanism for placing images in a paragraph.

But...

HTML, Word... are simpler to use. Be aware.

Sveta uses Photoshop interactively for a.o. coloring. MetaPost and ilks allow for coloring in a workflow. I don't know how to achieve by MetaPost the effects Sveta can do in Photoshop.

Libraries for macros and pictures are necessary for reusing parts. AnyT_EX is a preprocessor of T_EX! T_EX itself has a built-in preprocessor called macro expander. MetaPost is a preprocessor of PostScript, and even can produce SVG, with the MetaFont ingenuities inherited. So at the basis is plain T_EX and PS.

A nicety in the table chapter of PWT, next to the wealth of examples similar to those given in the T_EXbook, and some more, is a little spreadsheet functionality, which can be used in (budget) table markup, to enhance data integrity. It automates addition or subtraction of table entries, which is not that trivial because T_EX does not come with the ordinary calculator functionalities.¹⁰

This is similar to my plea of long ago in the IFIPWG2.5: write portable numerical (library) algorithms in the lowest higher-level language, FORTRAN, for use in all other higher level languages. Moreover those FORTRAN algorithms could be highly optimized, for example the matrix routines of $O(n^3)$ complexity, with n the order of the matrix. At the computer center of the Groningen University we spelled out the interfacing from PASCAL, ALGOL(68), and Simula to FORTRAN, and supplied users with this information, on-line, together with the availability of FORTRAN numerical libraries. We even contracted CDC to adapt their ALGOL68 compiler towards

FORTRAN interfacing. Realistically, I expect that my plea will be partially obeyed ... again.

Data integrity is all important. I paid attention to data integrity in among others my bridge macros, where once a card is played, it can no longer show up in the diagrams. Data integrity was also on my mind when I created macros for typesetting crosswords.

BTW, in the suprematistic Lozenge below Mondriaan was nearly right in dividing the sides according to the Golden Ratio. This Lozenge of 1925 was the last in a series ending with this minimal one. Others have some colored parts or more lines.

P. Mondriaan
Lozenge
Composition
with two lines

The bundling of the various macros, pictures, references, tools gave rise to my BLUE collection, nicknamed after Ben Lee User in the T_EXbook.

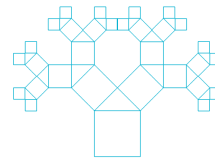
Like Knuth's plain etc macros and Berry's eT_EX macros, my BLUE collection is composed of parts to be reused in anyT_EX. Within BLUE what is needed can be obtained by selective loading, similar to retrieval from a database. One only loads what is needed! Even simpler, when not in the context of BLUE, is just to copy what you need, as I did for this note, see later.

But...

that is not simple enough, a library with ready to use modules is what we need.

From the absence of my BLUE in the FAQs of the UKTUG, the T_EX archives, and the T_EX collection DVD, I presume that the T_EX community missed the reuse-of-parts aspect of BLUE. Partly true, as I became aware at the conference: the bottleneck is the copyright.

In T_EX education language fluency is a prerequisite. Teach how to typeset ordinary language, technical jargon, e.g. mathematics, next to awareness of similar tools, the pro and cons of competitors.



Insight

Characteristics of insight are

- Abstraction
- Separations of Concerns, SoC
- Parameterization
- To foresee the future
- To use T_EX&Co
- To adhere Minimal Markup, Suprematism
- To use Paradigms
- To reuse parts

Dijkstra in the past mentioned that abstraction is our only mental tool to master complexity. As computer users we abstract all the time.

pdfT_EX violates SoC adage. I experience a retrograde. Inserting a color for example does not obey the scope rules in pdfT_EX. So the goodies of the past are annihilated. Why not keep the past achievements upright? I understand that we don't have the broad oversight Knuth had, and sin against all kinds of situations we don't foresee. Add whatever you want.

But...

without disturbing the achievements of the past, please. It is no good that a casual user like me is used as a guinea-pig. Test your materials thoroughly before releasing, please. Adhere to the practice of β -releases, such that a casual user is warned.

MetaFont is the big example of parameterization where each glyph is characterized by dozens of parameters. To handle gracefully related parameters Knuth invented the suffix concept, as far as I understand it is a unification of the common index and the PASCAL record, in the minimal style. In creating pictures it is a good habit to parameterize for the size, because then we can selectively scale. The line thickness is not altered if the size is changed. By blunt overall scaling the line thickness also changes, which might not be what you want.

Knuth forecasted the future by saying that he would use T_EX a hundred years after the birth of T_EX with the same quality as that of the beginning days.

Using paradigms in macro writing will increase readability.

One needs time and courage to invest in the use of plain T_EX and MetaPost, which will serve a lifetime, and will enrich your insight. Learning just a little bit of PostScript will not harm. You will be surprised by what you can achieve by it, with Adobe Photoshop as finishing touch for (interactive) coloring or removing hidden lines.

On the other hand if the majority of the T_EX community spends time and energy on L^AT_EX, ConT_EXt, LuaT_EX, in general on successors of T_EX, ... it is hard to stay with plain T_EX, to stay with Knuth, which means without development.

However, if one thinks a little deeper, it is an ill-posed rhetorical suggestion.

T_EX is a fixed point, only the environment changes

Adaptation to PS is for example taken care of by the driver dvi(2)ps and ilks, and pdf output can be obtained by Distiller or Acrobat (not tested though by me for a document), or just one of the various pstopdfs. T_EX commands for handling colors and inclusion of graphics are dictated by the drivers and have to be included in \specials. I have no problems at all to leave MetaFont for MetaPost, because... well, again an ill-posed suggestion. I don't leave MetaFont, I just don't use it for graphics any longer, I'm not a font designer. Well, ... again partially true: I'll continue to use MetaFont as my poor man's limited MetaPost previewer.

MetaFont and MetaPost have different purposes: the first one is aimed at bitmap font design, the second at creating PS graphics for inclusion in AnyT_EX or troff. The MetaFontbook is still needed because of the unusual concepts suffix, vardef, primarydef, secondarydef, and tertiarydef, which MetaPost has taken over from MetaFont, and which I don't grasp completely, yet.

A middle road is provided by ConT_EXt, which also comes with a wealth of documentation and is actively supported by its author Hans Hagen and colleagues, for example Taco Hoekwater.

A real breakthrough would be an interactive T_EX, which I would use immediately.

One can look upon this as what Apple did. They adopted UNIX as the underlying OS, and built their nice GUI on top. Comes T_EXshop close?

Minimal markup As said in my PWT guide, I favor to start with just text in a WYSIWYG way. Once you have your contents more or less right, have it spell-checked, and only then insert as few markup tags as possible. The result is what I call a Minimal Marked up script.

But...

do you have the patience to work along these lines? In reality this is my logical way of working. In practice I insert already markup once I have a reasonable version. Or, do you favor to rush into code and start with \begindocument...etc, without having written a word of the contents yet? Marvin Minsky drew attention to this already in his Turing Award lecture of long ago 'Form versus Content.' This approach, to markup at the end and use as little as possible of T_EX, is next best to WYSIWYG-T_EX, and my main reason to practise Minimal Markup.

Below the essentials of my Minimal Marked up script, obeying the 20%-80% adage, for this paper is given.

```
\input adhocmacros
\author ...
\abstract ...
\keywords ...
\head Script
...
\subhead
```

```
...
\jpgD ...
... \ftn ...
\bye
```

In order to mark up in the above spirit, I have borrowed from BLUe the following macros¹¹

```
\def\keywords#1\par{...}
\def\abstract#1\par{...}
\def\((sub)sub)head#1\par{...}
\def\ftn#1{...}%#1 footnote tekst
\def\beginverbatim \def\endverbatim
\def\beginquote \def\endquote
```

while \jpgD was just created for the occasion. Handy is the \ftn macro, which takes care of the automatic numbering of the footnotes. While working on this note, which a.o. emphasizes the use of Minimal Markup, I adapted the \ftn macro, such that the curly braces around the footnote text are no longer needed: just end the footnote by a blank line or a \par, implicit or explicit.

Also convenient is the functionality of a Mini-ToC. For the latter all you need is to insert

```
%In \head
\immediate\write\toc{#1}
%In subhead
\immediate\write\toc{\noexpand\quad#1}
%In subsubhead
\immediate\write\toc{\noexpand\qqquad#1}
```

Of course

```
\newwrite\toc
\immediate\openout\toc=\jobname.toc
```

must be supplied in the adhoc macros as well. Reuse on the fly!

But...

A L^AT_EXie would shrug shoulders, because (s)he has got it all already. True!

But...

at the loss of minimal markup. A BLUe user has both the macros and the minimal markup. A matter of choice, choose what you feel comfortable with.

If you like to concentrate on contents, clean scripts, abhor the curly braces mania, to err less and less, then Minimal Markup is for you.

Knuth's approach

What astonishes me most is that Knuth's plain.tex is not embraced by the majority. His basic approach should be taught, because in T_EX, well in automated digital typesetting, there are so many subtle things, where

you will stumble upon sooner or later, which cannot be shielded away from you by AnyT_EX, completely. Just pushing the buttons—inserting by an IDE prompted markup tags—is not enough.

How come that users did not adopt Knuth's plain.tex? Is it impatience, because mastering the T_EXbook with plain.tex embodied takes time, and much more when not guided by a skilful teacher?

History has it, that first gains were preferred by adopting L^AT_EX, which dares to explain less, keeps you unaware, which emphasizes the structure of documents, though not so rigorous as SGML, in a time when structuring whatever was en vogue. I'm not saying that structuring is wrong, not at all.

But...

one should not overdo it, one should not suggest it is the one and only. Keep eyes open, be on the alert for other aspects. On the other hand L^AT_EX comes with a lot of packages, nowadays.

When the minimal markup attitude is adopted, one does not need that many markup instructions! The structure is already there, in what you have to say, no need to overdo it. For this note I used basically a handful of structural macros, well... a few more, to be replaced by the ones used by the editor.

Knuth was right from the very beginning, though... not completely!

John Plaice commented on my presentation

'... that it was not possible to praise Dijkstra and Knuth in the same sentence as the two held completely opposite points of view with respect to programming languages. When Dijkstra published his 'Go to considered harmful'(CACM 11(3):147-148, 1968), Knuth defended the goto statement with his 'Structured Programming with go to Statements' (Computing Surveys 6(4):261-301, 1974).

According to Plaice, Knuth consistently supported the use of low-level languages for programming. In writing his TAOCP series of books, Knuth used his own assembler, the MIX, which he updated to a RISC machine, the MMIX, for the latest edition. His T_EX: The Program book regularly makes use of gotos. The register model used in T_EX programming is that of an assembler and the syntax is COBOLish.

Knuth's T_EX macro language has been criticized publicly by Leslie Lamport, who stated that if he had known that T_EX would have survived for so long, that he would

have fought much more strongly with Knuth for him to create a more usable language.’
Fair enough! Especially easier languages.

But...
I don’t see what is against Knuth’s attitude. Just a different approach.

Remember...
There Is More Than One Way To Do It.
I appreciate the features of high-level structured programming, with no, well... little, but well-defined side-effects, like for example exception handlers.

But...
when I learned Algol68 at the time, I was much surprised, because it seemed to me one big side effect. For me Knuth¹² knows what he is talking about, and he like nobody else, produced marvelous errorless tools, so SuPerBe documented

S_{uper}uP_{leasant}to read eR_{Beyond}thoroughness^e

T_EX&MetaFont

Thirty years ago the twin T_EX&MF was born. T_EX is aimed at typesetting beautiful Math by computer. MF was developed for providing the needed (Computer Modern bitmap) fonts.

Knuth was apparently so convinced of the correctness of his programs that he would reward every reported bug in T_EX with a check of 1\$, to start with, and he would double the reward each time an error was reported. We all know the exponential growth behavior of this. In the Errors of T_EX, Software Prac&Exp 19,7 1989, 607–685, Knuth mentions all the errors he corrected, to begin with those found while debugging.

T_EX had the following limitations in its design, on purpose.

- No WYSIWYG
- No Color
- Poor Graphics
- No Pictures (inclusion)
- No Communicating Sequential Processes¹³

How to overcome?

Thanks to \special, PS, PDF, the drivers and pdf(Any)T_EX

we have the following ways out, in order to compensate for what we miss

Script $\xrightarrow{\text{T_EX}}$.dvi as default

Script $\xrightarrow{\text{T_EX}}$.dvi $\xrightarrow{\text{dvi2ps}}$.ps for color and graphics

Script $\xrightarrow{\text{T_EX}}$.dvi $\xrightarrow{\text{dvi2ps}}$.ps $\xrightarrow{\text{ps2pdf}}$.pdf

or directly, the popular one-step

Script $\xrightarrow{\text{pdf(Any)T_EX}}$.pdf

I favor the multi-step way, the 3rd, which is in the UNIX tradition of cooperating ‘little’ languages, where processes are ‘piped’ in a chain, and which adheres to the Separations of Concerns adage. With respect to T_EX&MetaPost we talk about well-designed and time-proven programs.

I don’t believe that one person, or even a group, can write a monolithic successor of T_EX, of the same quality as T_EX and the time-proven cooperating little languages like dvi(2)ps, MetaPost, Acrobat and ilks.

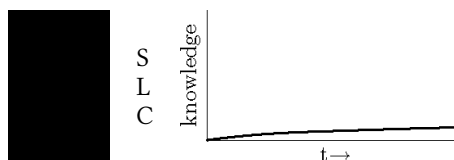
In the direct use of pdfT_EX I stumbled upon..., well... undocumented features?

Drawbacks of T_EX

It is interesting to read chapter 6 of the T_EXbook again about running T_EX. History! Is it? Still reality for plain T_EXies?

Next to the limiting choices made in the design of T_EX there are the following drawbacks

- T_EX SLC 120+% Energy¹⁴



- After so many years AnyT_EX&Co lack some sort of stability. Me, for example, I don’t have MetaPost running in an IDE. The T_EXnicCenter is not perfect, does not provide for menus suited for plain T_EX, too smart editor...
 - No GUI¹⁵
 - But...
 - T_EX&Co 99+%Quality
 - T_EX&Co gives you full control

Drawbacks of MetaFont

Nowadays, when we ask in T_EX for a T_EX-known font of different size, it is generated on the fly.

MetaPost, which sounds like a successor to MetaFont, was intended for creating PS graphics to be included in T_EX documents, and not for creating PS fonts, despite `&mfplain`, which is not enough. MetaFont's bitmap fonts are outdated, because of the significant memory it requires and because it is not scalable. The gap was filled in 2001 by Bogusław Jackowski et al. by releasing MetaType. Latin Modern is the PS successor of T_EX's native bitmap Computer Modern fonts. Work is underway for use of OpenType (Math) fonts in T_EX.

Literate programming

I like to characterize literate programming by

- Aims at error-free and documented programs
- Human logic oriented, not imposed by computer
- Programming and documentation are done simultaneously
- Relational instead of hierarchical

The computer science department of the Groningen University pays attention to program correctness issues, mainly for small programs, heavily biased by the loop invariance techniques of the 70-ies. No real-life approach suited for large programs like Knuth's literate programming approach, by which T_EX was implemented.

But...

There's More Than One Way To Do It¹⁶

Even at Stanford I could not find offerings for T_EX classes nor classes for literate programming.
No need? Still ahead of time?

I consider education in literate programming important, which should be provided by computer science departments in the first place.

T_EX Collection DVD

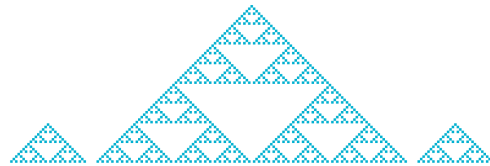
The DVD will lead you to the use of L^AT_EX or ConT_EXt. The IDE T_EXnicCenter allowed me to open projects and

process either

```
LATEX → .dvi, or
LATEX → .pdf, or
LATEX → .ps → .pdf
```

The Center does not provide buttons for processing plain T_EX with a format file of your own, at least that is not clear to me. I had to fool the system: opened a template, threw all that I did not need away and processed my minimal plain T_EX job as L^AT_EX!

Clumsy! Did I overlook something? The possibility to adapt T_EXnicCenter was on my mind for a couple of days. At last, I undauntedly defined an output profile and selected `pdfetex.exe`. Indeed, glory, my Hello World!\bye job worked, only the viewer Acrobat did not open automatically with the result file. The resulting .pdf file was stored in the same directory as the source file, so I could view it nonetheless. Not perfect as yet, because I would like to have the result opened in Acrobat automatically. Nevertheless, encouraging. I was surprised that `\magnification` did not work in `pdfetex`. I also stumbled upon that `\blue` defined as the appropriate `\pdfliteral` invoke, did not obey the scope rules. The editor in T_EXnicCenter is too smart: `\e` is changed into `é`, unless you insert a space. I reported this, no answer as yet.



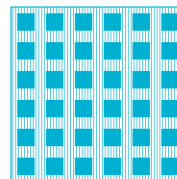
This paper asks among others to include in T_EXnicCenter also buttons for processing by Knuth's `plain.tex`. It would not harm to include as example the minimal T_EX job

```
Hello world!
\bye
```

or the T_EX classic `story.tex`, to demonstrate the workflow

```
TEX → dvi, pdf (or ps).
```

View buttons for either .dvi, .ps or .pdf results are nice.



I must confess that I was put off by the \TeX Collection after roughly a decade of my \TeX inactivity. The no longer maintained 4All \TeX CD of 1999 allowed me to \TeX this paper under Vista(!) with a handful of layout (preprint) macros, to be replaced by the macros of the editor of the proceedings. The outdated 4All \TeX CD contains a lot of interesting macros. Ingenious is the refresh option by just clicking the preview window in `windvi`, which alas, does not work properly under Vista. For pdf as output I had to get familiar with \TeX nicCenter. This is in contrast with for example Adobe: when you order software such as the Creative Suite, it is turnkey and works.

\TeX ing Paradigms Master Class

The \TeX arcane has become complex, very complex, and in my opinion too complex, if not for the number of languages one has know, as reported by Marek Ryćko at the Bacho \TeX 2009.

But...

in the spirit of UNIX, or LINUX as you wish, many little languages are unavoidable.

I favor to simplify. Educate the ins-and-outs of `plain` as basis, as a vehicle for digital typography, with a wink to `manmac` when real-life book production is at stake. A beginners' course on how to use \TeX is not necessary, because of the excellent tutorials, and the \TeX Collection installation (active) PDF document in your language to get \LaTeX , `pro \TeX t` or `Con \TeX t` running. How to run MetaPost is not yet provided for, did I miss something?

As already proposed a decade ago, I favor a class on minimal markup, on macro writing paradigms in plain \TeX , which I would propose nowadays as a Master Class, not on esoterics, but on macros we all need now and then, which are full of details worthwhile to know. In my macros you will not find 15 `\expandafters` in a row. Triads? Yes!

The prerequisite is that participants are Any \TeX users, who just want to broaden their understanding, who want to gain a confident way of robust macro writing, who like documents to be embellished by minimal markup.

This Master Class I would organize via the internet, although a \TeX course via the internet is not new. Times have changed. The teacher, or conductor, does not have to be the most knowledgeable \TeX ie, just the coordinator, like a chairman in a meeting. Attendees and coordinator commit themselves for a year or so, and have once a month a multiple participants session on the internet, with as many in between contacts as one can handle. The coordinator provides for exercises, where ideas to be worked out can come from participants too. In order to secure commitment the course is not for free, and the coordinator, under contract, is paid. A user group, for example NTG, backs it up and warrants continuity, for example with respect to the coordinator. For such a \TeX ing Paradigms activity I have material of 10 years ago as starting point, but would like to see added sessions on

- the use of the various `\last<...>`s,
- virtual fonts,
- active documents, read hypertexts, and
- \TeX with calls to MetaPost on the fly.

My old headings paper will be revised with emphasis on the three generations of headings already

1. Just as in plain \TeX
2. Provide for running heads in the headline and provide for a ToC creation, like in `manmac`
3. Provide for hypertext cross-referencing links and PDF bookmarks.

Of course, we might look into the CTAN for examples from the community.

The gain for participants is to master paradigms, to acquire a robust and minimal \TeX macro writing technique, just like Knuth. Absolute necessary is that the \TeX community backs up such a Master Class, by providing turnkey, mutual communicating \TeX &MetaPost on the fly, distributed for example via the \TeX Collection DVD. Although I am not in a good health, and have already said good-bye to most of my \TeX materials, I am available to conduct such a class, provided there is a stand-in, and turnkey \TeX \leftrightarrow MetaPost IDEs for PCs.

Another dream of me is a (hyperbolic) geometry class supported by MetaPost as tool.

M.C. Escher
Limit Circle III

T_EXing Paradigms beneficial?

In the sequel I will argue why a Master Class, or is it a sort of internet workshop, for T_EXing paradigms is beneficial.

It would be great if one could write a macro `\jpg` just at the moment one needs it. Reality is, I must confess, that it is not yet, well nearly, the case for me. Knuth, I was told in the past, can just do that.

In 1994 I published a note on the extension of plains `\item` macro in MAPS to provide for automatic numbering.

After my period of T_EX inertia I looked at it again, and I was much surprised. I missed the following considerations, also in the revision of 1996:

1. I missed the criterion that its use should be similar, and then I mean really similar, to Knuth's `\item`, with its minimal markup.
2. I missed the reasoning why it is useful to supply such a macro. Would not just the straightforward markup


```
\item1 text1
\item2 text2
etc
```

 make the need for a macro `\nitem` superfluous?
3. I overlooked that it was all about: a list of labeled indented paragraphs, each paragraph as usual ended by a `\par`—or the synonym `\endgraf`—inserted by either `\item`, `\itemitem`, `\smallbreak`, ... `\bigbreak`, ..., or implicitly by just the minimal markup of a blank line!

The point is: a good teacher would have drawn my attention to the ingenuity of Knuth's minimal markup, and strongly suggested not to write such a macro. Moreover, he would argue that there is hardly a need for it. MAPS was not reviewed, so neither a referee was in sight. Self-study is not enough, one needs guidance by a master. Little, first knowledge, which is usually acquired by self-study, is dangerous.

ConT_EXt and L^AT_EX provide for automatically numbered lists.

But...

within an environment, which is a clear and secure but different approach. It does not strive after utmost minimal markup.

If you have to mark up a document with long lists of numbered paragraphs a minimal `\nitem` macro, similar in use as `\item`, can be handy. I would propose the following `\nitem` nowadays, where I took care of my perceived impossibility at the time by redefining `\par` at an appropriate local place. The sequence of numbered paragraphs, to be marked up by `\nitems` is enveloped by a group behind the scenes, with the advantage that one can stay ignorant of the hidden counter.

```
\newcount\itemcnt
\def\nitem{\begingroup
  \def\par{\endgroup\endgraf}
  \def\nitem{\advance\itemcnt1
    \item{\the\itemcnt}}%
  \nitem}
%\par has to be replaced by \endgraf
\def\item{\endgraf\hang\textindent}
\def\itemitem{\endgraf\indent
  \hangindent2\parindent \textindent}
```

Another, maybe overlooked¹⁷ nicety is to have a `\ftn` macro, which maintains and uses a footnote counter. The user can just supply the contents of the footnote, does not have to worry about the (hidden) counter. My recent created Minimal Markup variant does not need curly braces around the (1-paragraph) footnote text, just end the text by a blank line. No

```
\futurelet\next\fo@t
```

needed.

If not convinced by my arguments a Master Class is beneficial by definition.

Examples of macro writing

To show you what I have on my mind in macro writing I have supplied a few macros.

Tough exercise

Glance at my recent solution of the T_EXbook tough exercise 11.5, which is more clear and more direct than the one given in the T_EXbook, IMHO, illustrating the

First-In-First-Out paradigm, as published in MAPS92.2 revised 1995, titled FIFO and LIFO sing the BLUEs--Got it? To end recursion a (classical) sentinel is appended and macro tokens are gobbled, the latter instead of Knuth's multiple use of `\next`. The assignment inhibits processing in the mouth, which in general I do not consider that relevant. This gobbling up of macro text in the mouth I use abundantly, e.g. in my \TeX macro for quicksort, as published in MAPS96.2. It also shows that sometimes we have to test for spaces and that sometimes the `%`-character is mandatory, especially when inadvertent spaces turn you down.



My current solution reads

```
\def\fifo#1{\ifx\ofif#1\ofif
  \else \ifx\space#1\space
    \else\blankbox{#1}%
    \fi
  \fi
  \fifo}
\def\ofif#1\fifo{\fi}
%
\def\blankbox#1{\setbox0=\hbox{#1}
  \hbox{\lower\dp0
    \vbox{\hrule
      \hbox{\vrule\phantom{#1}\vrule}
      \hrule}}}
%
\def\demobox#1{\leavevmode\fifo#1\ofif}
%with use
\demobox{My case rests.
Have fun and all the best.}
```

Is the use of `\ifx` essential, or could `\if`, respectively `\ifcat`, have been used?

Earlier, I had a solution where I read the line word by word using a space as parameter separator, circumventing explicit testing for spaces. So, at least three variants are available for discussing the pro-and-cons. I am curious for GUST's approach, because they have the bounding boxes of each character in GUST as part of their logo. Undoubtedly ingenious.

Knuth uses the functionality in as well the \TeX book ch18, to illustrate his explanation of the transformation of a math formula specification into a sequence of (boxed) atoms, as the MetaFontbook ch12 on Boxes. Reuse, aha!

Wind macros

Despite the powerful MetaPost, I will talk for the first time about my plain \TeX turtle line drawing graphics macros of ≈ 13 years ago, which I have used for a variant solution of the \TeX book exercise 22.14 (see later), but also for drawing some fractals, e.g. my favorite the Binary

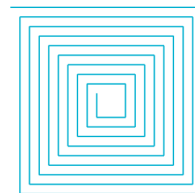
tree and the H-fractal (to be introduced later), a square spiral, and a Pythagorean tree, as well as a variant of the Sierpinsky triangle given at the end, as application of Knuth's dragon figures approach in appendix D of the \TeX book. The included smileys are new, they replace the old ones in `pic.dat`, because like Hans Hagen I guess, I'll do all my graphics in MetaPost or PS from now on. Revision, aha!

The wind macros can still be handy for sketches in \TeX alone, although I programmed already some fractals in PS directly. If time permits I might finish my fractals note, with the graphics in PS and MP.

`\N`, `\E`, `\S`, and `\W`, draw a line element from the point $(\backslash x, \backslash y)$ of size as supplied by the argument in the direction North, East, South, or West, respectively. The line element is encapsulated in a box of size 0, meaning the drawing does not change the reference point.

```
\def\N#1{\xy{\kern-.5\linethickness
  \vbox to0pt{\vss
    \hrule height#1\unitlength
    width\linethickness}}}%
\advance\y#1\unitlength}
%
\def\S#1{\advance\y-#1\unitlength{\N{#1}}}
%
%E and \W read similar, see my
%Paradigms: the winds and halfwinds. MAPS 96.1
%
\def\xy#1{%Function: place #1 at \x, \y
  \vbox to0pt{\kern-\y
    \hbox to0pt{\kern\backslash x#1\hss}\vss}}
```

`\xy` is similar to Knuth's `\point` macro of Appendix D of the \TeX book.



A straight application of the wind macros is the above shown (inward) square spiral, which is drawn by the macro listed below, where the number of windings is supplied in the counter `\k`, and the coordinates $(\backslash x, \backslash y)$ contain the starting point for the drawing. Note that during the drawing one has not to be aware of the coordinates, they are implicit. In order to display a centralized figure supply

```
\x = -\k * \unitlength
\y = \k * \unitlength
and enclose it in a box of height, width and depth
.5\k * \unitlength.
```

```
\def\inwardspiral{{\offinterlineskip
\loop\E{\the\k}\advance\k-1
  \S{\the\k}\advance\k-1
  \W{\the\k}\advance\k-1
  \N{\the\k}\advance\k-1
\ifnum\k>4 \repeat}}
```

Contest

I needed for the graphics inclusion in this paper, and in the slides, a minimal markup macro `\jpg`.

During my presentation I launched a minimal markup problem. I wanted to replace the following markup with optional `width... height...`

```
$$\pdfximage height..width...
  {filename.jpg}
\pdfrefximage\pdflastximage$$
```

by either the minimal markup

```
\jpgD filename
or
\jpgD width... filename
or
\jpgD height... filename
or
\jpgD height... width... filename
```

No square brackets, no curly braces, and even no explicit file extension, because it is already in the macro name.

I challenged the audience to write such a macro. There would be two winners one appointed by me and one by the audience.¹⁸

And... the winner is... Péter Szabó, by me and by the audience, with the following solution

```
\def\jpgfilename#1 {%
\egroup %end \setbox0\vbox
$$\pdfximage%
\ifdim\wd0=0pt\else width\wd0\fi
\ifdim\ht0=0pt\else height\ht0\fi
{#1.jpg}%
\pdfrefximage\pdflastximage$$
\endgroup}

\def\jpgD{%
\begingroup
\setbox0\vbox\bgrou
\hsize0pt \parindent0pt
\everypar{\jpgfilename}%
\hrule height0pt }
```

Elegant, to associate the optional parameter with a `\hrule` and measure it by putting it in a box. Elegant it is.

But...

I must confess, it took me some time to understand it. A

Master Class would have been beneficial for me, to learn to understand these kinds of macros more quickly 😊.

The near winner was Bernd Raichle with a thorough, straight-forward solution, not avoiding the pitfall, and which is a bit too long to include in this note.

But...

it shows a superb, elaborate parsing technique looking for `width... height...` and then take appropriate action.

Post conference Phil Taylor came up with a near, but intriguing solution and I myself also boiled up one. Both are supplied in the Appendix I, because much can be learned from them. Both neglect the (unintended) pitfall to concentrate on the parsing of the optional parameters. Phil and I just pass on the optional parameters, if any, to `\pdfximage`.

2D Graphics

Line drawings with incremental difficulties are included and discussed, such as

- straight line drawings by the wind macros and by `gkppic` in plain T_EX, and PS, such as fractals and flowcharts
- graphics with curved lines by PS and MP, such as graphics composed of oblique lines (and spurious envelopes), circles, and general curves, to be drawn by splines.

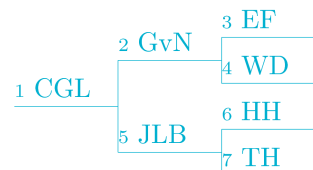
Binary tree

I consider my binary tree macro as very beautiful, because it can be programmed so nicely in T_EX or PS, and has such interesting applications. I consider it much in the spirit of Knuth's Turing award lecture *Computer Programming as an Art*.

```
\def\bintree{\E{\the\kk}%
  \ifnum\kk=2 \eertnib\fi
  \divide\kk2 {\N{\the\kk}\bintree}%
  \S{\the\kk}\bintree}%
\def\eertnib##1\bintree{\fi}%terminator
```

This mean and lean macro from the past can be adapted to your needs.

The above `\bintree` I used for a Turtle graphics, non-`\alignment` of the T_EXbook exercise 22.14, which reflects the binary structure. En-passant, NTG's VIPs replace the original names given in the T_EXbook.



The previous drawing is obtained via

```
%labels in preorder
\def\1{CGL}
\def\2{GvN}\def\5{JLB}
\def\3{EF}\def\4{WD}
\def\6{HH}\def\7{TH}
%
$$\unitlength2ex\kk8 \chartpic$$
%
%with adaptation to insert the leaves
%
\let\Eold\E
\def\E#1{\global\advance\k1
\xytxt{ \csname\the\k\endcsname$_\the\k$}
\Eold8}}
```

Remarks. The (educational) indices at the nodes are inserted to identify the nodes, to make the (order of) traversal explicit. The replacement text of \1 will be placed at node 1, etcetera. Adding the leaves, the initials, is done by adapting the \E macro and invoking \xytxt, which places text at the point (x, y). \chartpic encapsulates the Binary Tree picture in a box of appropriate size. See my earlier more elaborate note in MAPS96.1.

The variant PS code of the binary tree macro reads

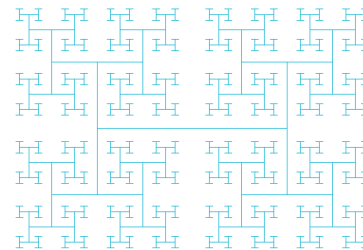
```
!PS -Bintree, cgl~1995-
%%BoundingBox: 0 0 600 600
/Bintree{/k exch def drawN
/k k 2 div def
k 1 gt {%
gsave drawE k Bintree grestore
drawW k Bintree}if
/k k 2 mul def}def %end BT
/drawN{0 k rlineto currentpoint
stroke translate
0 0 moveto}def
/drawE{k 0 rlineto
currentpoint stroke translate
0 0 moveto}def
/drawW{k neg 0 rlineto
currentpoint stroke translate
0 0 moveto}def
200 400 moveto 1 setlinewidth
.5 .5 scale 128 Bintree
showpage
```

I hope you will experiment with my binary tree codes, and please let me know if you have some nice, mean and lean use for it.

A more complicated use of the wind macros is the H-fractal macro as given below

```
\def\hfractalpic{%Size(2,1)*2\kk\unitlength
\def\hf{\ifnum\level>0
{\nxt1\hf}\nxt3\expandafter\hf
```

```
\fi}%
\def\nxt##1{\advance\dir##1
\ifnum3<\dir\advance\dir-4 \fi
\ifcase\the\dir \N{\the\kk}%
\or \E{\the\kk}%
\or \S{\the\kk}%
\or \W{\the\kk}%
\fi
\multiply\kk17 \divide\kk24
\advance\level-1 }%
\dir=0 \hf}%end hfractalpic
```



My PS variant is even more concise and reads

```
!PS -H-fractal cgl aug2009-
%%BoundingBox: 0 0 600 600
/Hfractal{/k exch def
gsave draw
/k k 2 mul 3 div def
k 1 gt {90 rotate k Hfractal
-180 rotate k Hfractal}
if
/k k 3 mul 2 div def
grestore}def %end Hfractal
/draw{0 k rlineto
currentpoint stroke translate
0 0 moveto}def
%
300 0 moveto 1 setlinewidth
.3 .3 scale
512 Hfractal
showpage
```

With respect to graphics I favour PostScript, and why not write yourself a little bit of mean and lean PostScript? The above macros in \TeX can be useful for sketches and fractals.

But... when linethickness becomes noticeable, they suffer from a severe disadvantage of ugly line joinings like the notch

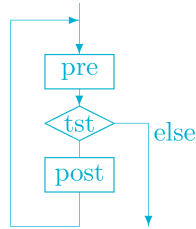


In MetaPost these notches can be circumvented by using suitable pens. Using in PostScript appropriate values for

setlinejoin, setlinecap, or the use of closepaths for contours, will help you out.

Flowchart

An example of the use of gkppic macros is the following diagram for the loop.



The code I borrowed from BLUE's pic.dat, adapted for the occasion with inserting \bluec and \bluel. Not nice, so another reason for doing it all in MetaPost.

```

\def\blueflowchartlooppic%
{\bgroup\unitlength=3ex%3.8ex
 \xoffset{- .5}\yoffset{- .3}%
 \xdim{5}\ydim{7.5}%
 \beginpicture
 %\ifmarkorigin\put(0,0)\markorigin\fi
 \put(0,0){\line(1,0){2}}%
 \put(0,0){\line(0,1){6}}%
 \put(0,6){\bluec\vector(1,0){2}}%
 \put(2,6.5){\bluec\vector(0,-1){1.5}}%
 \put(1,4){\framebox(2,1){\bluec pre}}%
 \put(2,4){\bluec\vector(0,-1){.5}}%
 %\put(2,3){\rhombus(2,1){tst}}%
 \put(1,3){\bluec\line(2,1){1}} %lu
 \put(1,3){\bluec\line(2,-1){1}} %ll
 \put(3,3){\bluec\line(-2,1){1}} %ru
 \put(3,3){\bluec\line(-2,-1){1}} %rl
 \put(2,3){\makebox(0,0){\bluec tst}}%
 %
 \put(2,2.5){\line(0,-1){0.5}}%
 \put(1,1){\framebox(2,1){\bluec post}}%
 \put(2,1){\line(0,-1){1}}%
 %
 \put(3,3){\line(1,0){1}}%
 \put(4,3){\vector(0,-1){3}}%
 \put(4,2.5){\kern2pt{\bluec else}}%
 \endpicture\egroup%
}% end blueflowchartlooppic

```

Before the invoke I defined \bluel as well as \bluec and used \bluec in the definition and \bluel before the invoke. Not so nice, but imposed by PDF.

Disadvantages of the gkppic macros and L^AT_EX picture environment is that coloring is tricky when using \pdf(Any)TeX: elements of a font are colored by

supplying **k** to \pdfliteral and lines or fills are colored by supplying **K** to \pdfliteral. Moreover, one has to put the pictures drawn by the wind macros in a box of appropriate size, sort of Bounding Box, to ensure space. A nuisance.

```

\def\bluec{\pdfliteral{1 0 0 0 k}}
%versus
\def\bluel{\pdfliteral{1 0 0 0 K}}

```

When we use the multi-step processing line via PS we don't have that disadvantage. Below a test example.

```

pretext
\special{ps: 0 0 1 setrgbcolor}%blue
abc
\hrule
def
\special{ps: 0 0 0 setrgbcolor}%black
posttext
\bye

```

Process the .dvi via dvips and view the .ps, to verify the result. In T_EXnicCenter I have added the Output Profile e_TX→PS→PDF. Convenient.

Note the explicit switching back to black, because the T_EX scope rule is not obeyed, of course. Another way to prevent that the rest will appear in blue, is to insert gsave in the first and grestore in the second \special.

My MP code for the flowchart which made use of Hobby's boxes.mp.

```

input boxes.mp;
prologues:=3;
%outputtemplate:="%j-%3c.eps";
beginfig(0)
boxit.boxa (btex \hbox to 60pt%
 {\strut\hss pre\hss}etex);
boxit.boxb (btex \hbox to 60pt%
 {\strut\hss tst\hss}etex);
boxit.boxc (btex \hbox to 60pt%
 {\strut\hss post\hss}etex);
boxa.c=(100,180);
boxb.c=(100,140);
boxc.c=(100,100);
boxa.dx=boxb.dx=boxc.dx=5;
boxa.dy=boxb.dy=boxc.dy=5;
drawoptions(withcolor blue);
drawboxed (boxa, boxc);
%draw contents of b and the diamond shape
draw pic.boxb;
draw boxb.w--boxb.n--boxb.e--boxb.s--cycle;
%The arrows
drawarrow boxa.s--boxb.n;
drawarrow boxb.s--boxc.n;
z1-boxb.e=z2-boxc.e=(20,0);

```

```
drawarrow boxb.e--z1--z2;
z3=boxb.w-(20,0);
z4=boxc.w-(20,0);
drawarrow boxc.w--z4--z3--boxb.w;
endfig
end
```

Note how the diamond diagram is obtained from the information created by the `boxit` invoke. Henderson's previewer does not allow the use of `boxes.mp`. If one can write the above flowchart with the use of `gkppic` macros, one could have coded it equally well direct in PS. To code in PS the generality incorporated in the `boxes.mp` is substantial more work, and unnecessary in view of the neat `boxes.mp` macros, which come with the PD MetaPost. In Goossens' et al. Graphics Companion the `metaobj` work of Roegel is mentioned as a generalization. To understand the `boxes.mp` macro is a good exercise in becoming familiar with the `suffix` concept, a unification of `index` and `record`, which Knuth introduced to handle conveniently the many related parameters in a glyph. By the way, the resulting PS is not standard PS, but named purified PostScript which is incorrect, because of the used `cmr` fonts. The coupling of the \TeX `cmr` fonts to PostScript fonts, an old problem, is done by the `prologues:=3;`. The name of the resulting output file can be composed to one's wishes by assigning an appropriate string to `outputtemplate`.¹⁹ The filename with extension `.eps` facilitates previewing via Acrobat. For previewing via `CSview4.9` it does not matter. The inclusion of the string `(100,100)` translate as value of `special` hinders previewing: the Bounding Box is incorrect, the calculation of the BB does not account for this PostScript inclusion. I have the impression that the PostScript code is just passed through to the output. Inserting the desired font via `special` would not work either. I found it convenient to name `boxes` beginning with `box...`, to avoid name clashes. See for more details and possibilities the (latest version of the) MetaPost manual.

If only the \TeX Live DVD would have provided installation directions for MetaPost.

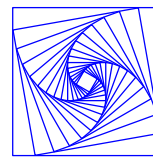
Oblique lines

When my youngest daughter was at school, she produced the following object created by 'oblique lines' with spurious envelopes (left). My emulated result in MP is at the right, and created just for this note.

The curves, not quarter circles by the way, suggested by the oblique lines which connect points on the sides, are spurious. Below I have included the MP code.

```
beginfig(1); numeric s; s=5cm;
path l, u, cl, cr;
%...
cl=(-s,-s){up}..(-.5s,.5s)..{right}(s,s);
cr=(s,s){down}..(.5s,-.5s)..{left}(-s,-s);
for t= 0 step 0.5 until 20:
draw point t/10 of cl --
      point t/10 of cr withcolor blue;
endfor;
pickup pencircle scaled 3pt;
draw l..u--(s,-s)--cycle withcolor blue;
endfigure;
```

If we rotate and shrink appropriately we get the interesting figure



In Goossens' et al. The \LaTeX Graphics Companion this figure is shown as an example of a recursive object.²⁰

```
beginfig(0);
u=1cm;
drawoptions(withcolor blue);
for k=0 upto 15:
draw((u,-u)--(u,u)--(-u,u)--
      (-u,-u)--cycle)rotated 10k;
u:=.86u;
endfor
endfig;
```

These figures are a prelude to Gabo's 3D regular surfaces.

PostProcessing by Photoshop

My wife, Svetlana Morozova, 'shoot-shoot' post processed the PS flower (left) interactively via Photoshop into the nice colored flower (right).

The PS code for the flower is given on the next page.


```

%%!PS Flower. CGL june 96
%%BoundingBox: -25 -25 25 25
0 0 1 setrgbcolor
100 100 translate
/r 18 def
10{r 0 r 90 180 arc
   0 r r 270 360 arc
   36 rotate}repeat
stroke
showpage
%%EOF

```

Have a try in coloring it. For comparison I have included the MP code below.

```

beginfig(1);
r=18;
path p;
drawoptions(withcolor blue);
p= (0,0){up}...{right}(r,r){down}
   ...{left}cycle;
for i=1 upto 10:
draw p rotated 36i;
endfor
endfig
end

```

Note. In MP we don't have to translate the origin. The connection between the knots is tight, by three dots. Interesting is that PS can draw circles while MP approximates.

PostScript spaces

MF and MetaPost are nice graphical languages, with convenient and powerful high-level instructions. Postscript employs the notion of a 2D user space and a 2D device space, the latter can rotate. MetaPost has a path and a picture data structure, which can be rotated, well... transformed. Another difference is that PostScript is stack oriented and MetaPost enjoys a declarative language with a thorough syntaxes.

Negative from MetaPost is that the resulting PostScript code may not be concise and readable. Values of the MP variables are given in the resulting PS and not their symbolic names. Loops are unwinded.

Raw PostScript can be included via MetaPost's special, however. The best of both worlds?²¹

M.C. Escher
← Bolspiraal
CGL's
→ Sort of

Yin Yang

A neat timeless MetaPost code I borrowed from Hobby

```

beginfig(1);
u=1cm;
path p;
p = (-u, 0)..(0,-u)..(u,0);
fill p{up}..(0,0){-1,-2}..{up}cycle;
draw p..(0, u)..cycle;
endfig;

```

without the 'eyes'.²²

Below my enriched PostScript variant of Hobby's MetaPost code, for the real Yin Yang picture.

```

%!PS-Adobe- Yin Yang. cgl July 2009
%%BoundingBox: -25 -25 25 25
/R 25 def      /hR R 2 div def
/mR R neg def /mH hR neg def
/r R 5 div def /mr r neg def
/rcircle {translate % center on stack
          r 0 moveto 0 0 r 0 360 arc
}def
0 mR moveto 0 0 R 270 90 arc
          0 hR hR 90 270 arcn
          0 mH hR 90 270 arc

fill
R 0 moveto 0 0 R 0 360 arc
stroke
gsave 0 hR rcircle fill grestore
gsave 0 mH rcircle
      1 setgray fill
grestore

```

It differs from the MP code, because there is no direction specification in PostScript. Procrusting direction has to be done by control points in general, which are not needed in this special case. If the small discs are omitted—Hobby's original code—the PS code is not that much longer than the MP code. Orientation is immaterial.

A picture with a genuine use of control points is the Malbork window below.

Syntactic sugar? Yes, but the PS code can directly be used in documents, to be inserted by the `dvi(2)ps` driver, alas not directly by `pdf(Any)TeX`. Strange.

It is tempting to go for `.pdf` all the way and I was advised to convert PS pictures into PDF, which is easy via Acrobat, or the older Distiller, Illustrator, Photoshop?, or... No big deal.

However, I could not control the placement of `.pdf` pictures in `pdfTeX`.²³ For this paper, and my slides, I have converted all `.eps` pictures into `.jpg`.²⁴

The `.jpgs` could be integrated smoothly in the document by `pdfTeX`, and was the way of picture inclusion with `pdfTeX`, for `EuroTeX09`, because it worked and I think `.jpg` is more appropriate for pictures, despite its non-scalability without quality loss

But,...

maybe PDF, SVG or PNG is better, no hands-on with the latter two formats as yet. However, I believe— biased by the SoC principle and because I can include `.ps` pictures in my document— that the more-steps processing, via `dvi(2)ps` is better. I will explore that later.

Smiley

A simple example in MP of the use of pens is the following MP code for the smiley ☺

```
beginfig(1);
u=1cm;
color yellow; yellow=(1, 1, 0);
fill fullcircle scaled 20u
    withcolor blue;
pickup pencircle scaled 4u;
draw ( 4u, 4u) withcolor yellow;
draw (-4u, 4u) withcolor yellow;
pickup pencircle scaled 1.5u;
draw (-7u,-u){1,-10}..(0,-7u)
    ..{1,10}(7u,-u) withcolor yellow;
endfig;
```

Schröfers opart

Placement of (deformed) circles in a square has been done by the artist Schröfer in an opart way. My emulation follows.

```
%!PS -Schröfer's Opart cglNov1996-
%%BoundingBox: -190 -190 190 190
200 200 translate
/s 5 def %BB for 1 with s=5
/drawgc{gsave
  r c translate
  r abs 5 div s add
  c abs 5 div s add scale
  0 0 1 0 360 arc
  fill
grestore}def%end drawgc
%
/indices [30 21 14 9 5 2 0
  -2 -5 -9 -14 -21 -30] def
/Schröfer{/flipflop true def
indices{/r exch s mul def
  gsave indices{/c exch s mul def
    flipflop{drawgc}if
    /flipflop flipflop not def
  }forall
  grestore
}forall
-38 s mul dup moveto
0 76 s mul rlineto
76 s mul 0 rlineto
0 -76 s mul rlineto
closepath 5 setlinewidth stroke
}def%end Schröfer
gsave .5 .5 scale Schröfer
grestore
showpage
```

EuroTeX94 battleship logo

In order to illustrate the calculation of intersection points in PS I have borrowed the EuroTeX94 battleship logo.

In general we need MF or MP for solving (linear) equations within a graphics context. However, the calculation of the intersection point of 2 straight lines we learned already at highschool, albeit without pivoting strategy, for numerical best results. So, the knowledge for solving

2 equations in 2 unknowns in PS is there. I spent a little time on writing the PS code for it, \approx 15 years ago, just after the 1994 EuroT_EX, to write the logo in PS when MP was not yet in the public domain. I did not realize at the time that it was important, because people believe, have the prejudice, that we can't solve equations in PS, or at least they apparently think that we don't have a def for it. Indeed, we should create a library of .ps defs, or maybe it exists already somewhere?

What we miss so far is that we can't specify in PS the equations implicitly as we can in MF and MP. No big deal.

From the specified points on the stack in PS we can form the coefficients for the equations, leave these on the stack and solve the equations. No more than high school knowledge is required, and... a little endurance to solve the equations, as I did maybe some 30 years ago for the HP handheld calculator, which also uses a stack and Polish Reverse Notation.

The data in PS code reads

```
%Data
/p0{0 0}def
/p1{3 s mul 0}def
/p2{4.5 s mul 2 s mul}def
/p3{3 s mul s}def
/p4{-.75 s mul 2 s mul}def
/p5{p0 top p3 p4 intersect}def
/p6{p0 p1 mean top p3 p4 intersect}def
/p7{top p1 p3 p4 intersect}def
/p8{p2 p5 top p1 intersect}def
/p9{p8 dup 0 exch top p0 intersect}def
/top{2.5 s mul 3 s mul}def
```

To specify all the points I needed a PS def intersect for calculating the intersection point of 2 lines determined by 4 points.

Points $p_1 p_2 p_3 p_4 \rightarrow x y$

```
/p1{0 0}def /p2{10 0}def ...
%
/p {p1 p2 p3 p4 intersect}def
%
/intersect {%p1 p2 p3 p4 -> x y
makecoef 7 3 roll
makecoef
solveit}def %end intersect
```

```
%
/makecoef{%z1 z2 -> e a b
4 copy      %x1 y1 x2 y2 x1 y1 x2 y2
4 -1 roll mul
3 1 roll mul sub
5 1 roll 3 -1 roll sub
          %(y2x1-y1x2) x1 x2 y2-y1
3 1 roll sub%(y2x1-y1x2) y2-y1 x1-x2
}def %end makecoef
```

As last piece the definition of solveit

```
/solveit{%e a b f c d -> x y
%Equations: ax + by = e  p=pivot
%              cx + dy = f
%pivot handling %e a b f c d
1 index abs     %e a b f c d |c|
5 index abs     %e a b f c d |c| |a|
gt {6 3 roll} if %exchange 'equations'
%stack: e a b f c d or f c d e a b,
%first is in comments below
exch 4 index    %e a b f d c a
div             %e a b f d p
6 -1 roll dup 6 1 roll 3 1 roll
              %a e b f e d p
4 index exch    %a e b f e d b p
dup 4 1 roll    %a e b f e p d b p
mul sub        %a e b f e p (d-b.p)
4 1 roll mul sub exch div
%a e b (f-e.p)/(d-b.p) = a e b y
dup 5 1 roll mul sub exch div exch
}def %stack: x y
```

Finally, the drawing of the battleship

```
%Battleship
-2 s mul 0 translate
0 0 1 setrgbcolor
p0 moveto p1 lineto p2 lineto p3 lineto
                    p0 lineto closepath
p1 moveto p3 lineto p4 lineto p0 lineto
p5 moveto top lineto p6 lineto
p6 moveto top lineto p7 lineto
p2 moveto p8 lineto p4 moveto p9 lineto
stroke
```

Circle covered by circles

This example is included because it demonstrates that even in PS we can solve nonlinear equations.

Essential in this code is the definition of Bisect for zero finding of a nonlinear function.

```
/Bisect{%In 0<=l<u f(l)<0 f(u)>0
      %Out l<=d<=u u-l<eps f(l).f(u)<=0
}/fd f def
fd 0 lt {/l d def}
      {/u d def}ifelse
u l sub eps gt
  fd 0 ne and %l-u>0&f/=0
{Bisect}if
d}def %end Bisect
%
/l ...def /u ...def
/d{.5 l u add mul}def
/f{% f: d-->f(d)
  ... } def
```

For the complete code and the formulas for the mid-points of the circles see my Tiling note of the mid 90-ies.

2.5D Graphics

For drawing 3D objects in PS I discern the following spaces

- 2D PostScript Device Space
- 2D PostScript User Space

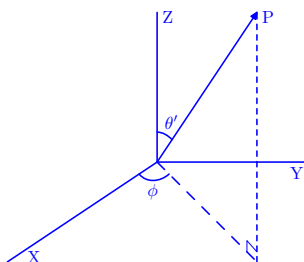
I added

- 3D User Space, the data
and
Project 3D US
onto 2D PostScript US

By 2.5D graphics I mean an image of a 3D object, displayed as 2D graphics, obtained from 3D data specifications and projection onto 2D.

The projection formula reads

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -\cos \phi & \sin \phi & \cos \theta \\ -\sin \phi \sin \theta & -\cos \phi \sin \theta & \sin \theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



The (full) transformation matrix can be understood as to be composed of 2 rotations: first around the z-axis and second around the transformed x-axis, such that the z-axis coincides with the view direction $OP \perp$ the new xy-plane. We can omit the 3rd, the transformed z-coordinate, because it is no longer visible in the (orthogonal) projection. The factorization of the projection matrix is

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -\sin \theta & \cos \theta \\ 0 & \cos \theta & -\sin \theta \end{pmatrix} \begin{pmatrix} -\cos \phi & \sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Coded in PS as a Point to Pair def, ptp, the projection formula reads

```
/ptp{/z exch def/y exch def/x exch def
x neg a cos mul y a sin mul add
x neg a sin mul b sin mul y neg a cos mul
  b sin mul add z b cos mul add}def
```

Later, in the MP code for Gabo's linearii, the MP vardef for Point to Pair is given.

Pyramids

Hobby in his 'A user's manual for MetaPost' draws a pyramid. Below I have drawn pyramids starting from 3D data as function of the viewing angles.

The PS code reads

```
%!PS-Pyramid in projection, cglau2009-
%%BoundingBox: 0 0 300 100
/ptp{/z exch def/y exch def/x exch def
x neg a cos mul y a sin mul add
x neg a sin mul b sin mul y neg a cos mul
  b sin mul add z b cos mul add}def
%
/r 20 def /hr r 2 div def
/z1{r neg r 0 ptp}def
/z2{r neg dup 0 ptp}def
/z3{r r neg 0 ptp}def
/z4{r r 0 ptp}def
/top{0 0 r 4 mul ptp}def
%
/pyramid{z1 moveto z2 lineto z3 lineto
  [2]1 setdash stroke
```

```

z3 moveto z4 lineto z1 lineto
top moveto z1 lineto
top moveto z3 lineto
top moveto z4 lineto stroke
top moveto z2 lineto
  [2]1 setdash stroke}def%end pyramid
%
30 300 translate
0 0 1 setrgbcolor%blue
1 setlinecap 1 setlinewidth
%
15 25 65{/a exch def
30 -20 10{/b exch def
      pyramid
      57 0 translate}for}for
showpage

```

Naum Gabo

← Lineari

→ Linearii

I also passed by his (temporarily nonworking) fountain in front of the St Thomas hospital opposite the Big Ben, on the other bank of the river Thames.



Escher's impossible cube

As student I was asked by my professor to (re)make Escher's²⁵ Impossible Cube.

I decomposed it into two parts, in timber, and put them together such that in projection the impossible cube was obtained. I photographed it and handed the photo to my professor.²⁶

I'm happy that after so many years, I had the guts to emulate the cubes.

I consider each corner of the (impossible, non-mathematical) cube as a cube itself, with its 8 corners as data points, which yields in total 64 data points. After projection I could connect the appropriate points and make the (impossible) cube. First, I did the erasing and adjusting in Photoshop, but a little later I drew the impossible cube in PS alone, which is not completely general as function of the viewing angles. A bit tedious. The code is too long and too tedious to be included here.

Gabo's constructive art

Long ago, I was still a student, I visited the Tate Gallery in London, and was captivated by the constructive art of Gabo,²⁷ especially his *Linearii* of the early 1940-ies.

In the fountain, the regular surface is formed by jets of water, and changes dynamically, because it rotates, due to the 'action is reaction' principle.

After many years, I all of a sudden had an idea of how to imitate this fountain in my garden, for my quarter circle pond. A very remote sort of imitation, but... funny. Too alternative to be included here, maybe on the slides for *BachTeX2010*, for fun.

With my youngest daughter I imitated Gabo's *linearii* in plastic.

In ≈ 1995 I emulated *lineari*, *linearii* in MF, and adapted the code towards MetaPost. For this conference I looked at the pictures again. Sveta and I adjusted them with thinner lines and colored them blue.

They came out more beautiful than before, even nicer than the photo's of the objects, IMHO. A matter of taste?

Naum Gabo
Lineari

Naum Gabo
Linearii

Of linearii I have included the MP code below

```

beginfig(1);
proofing:=1;
size=75;
path p[];
def pointtopair(expr x,y,z)=
(-x*cosd a + y*sind a,
-x*sind a * sind b -y*cosd a * sind b
+ z*cosd b)
enddef;
%
%Path construction
%
%basic path (the shape of the boundary)
%can be molded, can be constrained etc
p1:= (0,3size){right}..
{down}(1.1size,1.75size){down}..
(.35size,.75size)..(.175size,.375size)..
{left}origin;
%path with regular---nearly so---
%distributed points
n:=0;%number of points along the curve
p10:= point 0 of p1 hide(n:=n+1)..
for t:=1 upto 19: hide(n:=n+1)
point .05t of p1..endfor
point 1 of p1 hide(n:=n+1)..
for t:=1 upto 13: hide(n:=n+1)
point 1+t/14 of p1..endfor
point 2 of p1 hide(n:=n+1)..
for t:=1 upto 3: hide(n:=n+1)
point 2+t/4 of p1..endfor
point 3 of p1 hide(n:=n+1)..
for t:=1 upto 3: hide(n:=n+1)
point 3+t/4 of p1..endfor
origin;
%viewing angle parameters
b:=-10; a:=60;
%Project the nodes and create
%‘paths in space’ the boundaries
p100:= for k=0 upto n-1:
pointtopair(0,xpart(point k of p10),
ypart(point k of p10))..
endfor pointtopair(0,0,0);
p200:= for k=0 upto n-1:
pointtopair(xpart(point k of p10), 0,
ypart(point k of p10))..
endfor pointtopair(0,0,0);
p300:= for k=0 upto n-1:
pointtopair(0,-xpart(point n-k of p10),
3size-ypart(point n-k of p10))..
endfor pointtopair(0,0,0);
p400:= for k=0 upto n-1:
pointtopair(-xpart(point n-k of p10),
0, 3size-ypart(point n-k of p10))..
endfor pointtopair(0,0,0);
%
%Drawing
%
%MetaPost approach: black background
% and whitedraw
%Black background
fill (-1.5size,-size)--(-1.5size,5size)--
(1.5size,5size)--(1.5size,-size)--cycle;
%
%Below white drawing
drawoptions(withcolor white);
%
pickup pencircle scaled .5pt;
%Top ring and hang up (rope)
draw point 0 of p100..
point 0 of p100 + (0,.1size)..cycle;
draw point 0 of p100 + (0,.1size)..
point 0 of p100 + (0,3size);
%Draw boundary curves
draw p100; draw p200; draw p300; draw p400;
%
%Draw (partially hidden) regular surfaces
pickup pencircle scaled .1pt;
for k=0 step 1 until n:
draw point k of p200..point n-k of p300;
endfor
for k=0 upto n:
draw point k of p400..point n-k of p100;
endfor
%erase the ‘hidden’ parts of the lines
%erase fill p100..reverse p200..cycle;
%MetaPost might blackdraw this
%fill p100..reverse p200..cycle
% withcolor black;
%Front
pickup pencircle scaled .1pt;
draw p100; draw p200;
draw point 0 of p100--origin;

```

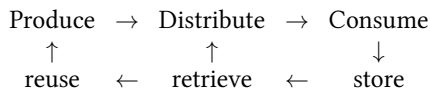
```
%
%Draw regular surface which is in sight
for k=0 step 1 until n:
  draw point k of p100..point n-k of p200;
endfor
%Clip to boundary, mod July 2009
clip currentpicture to (-1.5size,-size)--
  (-1.5size,5size)--
  (1.5size,5size)--(1.5size,-size)--
  cycle;
endfig;
end
```

Mathematically, I love the above included regular surfaces due to Gabo, because they are constructed from 1-dimensional data, the bounding curves in 3D. The necessary parameterized projection technique also facilitates animation by changing the viewing angles.

For the first time I emulated the real Gabo by not erasing the ‘hidden’ lines. In reality they are not hidden, because the object is made of transparent perspex. I lied a bit in the past, because when I did not erase the hidden lines the reverse video picture looked too much blurred by detail. For this conference I fine-tuned the picture with thinner lines and in blue, which looks OK to me.

Reuse

Sooner or later one arrives at the situation to organize the wealth of macros, pictures, references, tools and ilks for reuse. This gave rise to my BLUE collection. The idea in BLUE is that all the macros you use most of the time are bundled into a `blue.tex` kernel. The rest is split up into: tools, formats, pictures, references, addresses,... of which BLUE will reuse parts on the fly, unaware of the filing system of the computer. Reuse is a general important aspect, and ipso facto in the lifecycle of document parts.



With a monolithic collection you have it all, all the time. I decided to adhere to the kernel&modules adage, and to use only from a module what is needed, realized by a selective loading technique, similar to the one of M. Diaz, as mentioned in appendix D of the T_EXbook.

One might argue that this economy has become more and more irrelevant, because of the enormous increase of computer speed and memory, since the birth of T_EX. Partly true: sometimes parts conflict, e.g. one either formats a report or an article, and in general it is safe to avoid possible conflicts.

To illuminate this note, I have reused pictures be it from `pic.dat` or from separate PostScript files.

Sometimes reuse implies some extra work.

I also reused the commands included in `\loadtoc` macros together with `\pasteuptoc` for a mini-ToC to keep track of the structure while creating this note. En passant the Contents at the beginning was obtained.

This proceedings submission differs from the pre-proceedings one, because working on the slides gave feedback. The 2.5D GABO’s as well as the Escher Cube have earned a place for their own.

An invoke of the one-part `\bluepictures` followed by one or more `\picturenames` will load the indicated (T_EX) pictures and make them available under their names. At the time I did not construct a library of PostScript pictures, because I did not know how to supply these to `\epsfbox`. There is no `pspic.dat`, alas. If time permits I will think it over.

Another aspect is the search path of drivers, if only they looked into `TeX input` or `texmflocal`; where to put `pspic.dat`?

It is not so handy to put the pictures directory in the same place as the main document. I do not know how to add search paths.

If only `\pdfTeX` could handle PostScript...

As alternative to `\bluepictures` one can use the underlying two-part macros, sort of environment

```
\beginpictures
  \picturename1
  ...
\endpictures
```

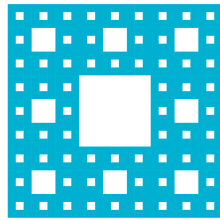
but, alas T_EX has no garbage collector, it would not save on memory, it only reduces the possibility of conflicts.

Similar structures hold for tools, formats, references, ...

In 2002 I worked on macros for the automatic generation of PDF bookmarks for the ((sub)sub)heading titles. It worked, even presented them at the EuroT_EX, if I’m not mistaken.

But...

I did not finish the macros, maybe I should. I noticed that in 2005 A. Heck did not provide for bookmarks in his MetaPost note published in MAPS, also available on his WWW site. Is there a need? Rhetorical question.



The above Sierpinski picture was done in \TeX with its pseudo filling via rules, also called **black boxes** by Knuth. \TeX lacks the filling of an arbitrary closed curve, if not for coloring it.²⁸ Hyperlinks were also invented long after the birth of \TeX . pdf \TeX makes that all possible.²⁹ I missed what extras e \TeX , or NTS as successors of \TeX have brought us. I hope to learn at this conference what Lua \TeX is all about.

For me plain \TeX mutual communicating with MetaPost, with PDF as result is sufficient, and even better when PS pictures can be included.

Maybe I can work around it by including PS in MetaPost with SVG or PDF out, or by the multi-step route via `dvi(2)ps`. It is so strange that the world outside has adopted PS and we don't in pdf(Any) \TeX .³⁰

3D metaPost?

It should not be too difficult to extend MetaPost with triples, (x, y, z) , in analogy with color, for 3D data. Transformations can then be defined by the 12-tuple $T_x, T_y, T_z, T_{xx}, T_{yy}, T_{zz}, T_{xy}, T_{xz}, T_{yz}, T_{yx}, T_{zx}, T_{zy}$. In matrix notation

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix} + \begin{pmatrix} T_{xx} & T_{yx} & T_{zx} \\ T_{xy} & T_{yy} & T_{zy} \\ T_{xz} & T_{yz} & T_{zz} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Conclusions

Whatever your tool in computer-assisted typesetting, a Master Class on macro writing in plain \TeX and MetaPost is worthwhile, along with discussing tools for similar and complementary tasks to increase awareness, insight and productivity.

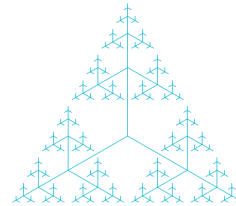
A course in literate programming would be great too.

Wishes

May my turtle graphics macros find a niche, and may my BLUe collection and Paradigms notes be saved from oblivion and kept alive via distribution on the \TeX Collection DVD.³¹

May the \TeX Collection maintainers, the \TeX nic-Center authors, and Jonathan Kew in his \TeX works,

support the plain \TeX and MetaPost users as well as the L \TeX and Con \TeX t users.



Hopes...

\TeX &Co world will

- Adopt Knuth's Plain & Adobe's PS
- Adhere to Paradigms: SoC...

I learned a lot at and after the conference.

\TeX ies, who have been out of it for a while, are well re-educated at an Euro \TeX meeting.

Lua \TeX and X \TeX are \TeX engines, which a.o. provide support for Unicode and OpenType fonts. New OpenType fonts for use in Lua \TeX and X \TeX are Latin Modern and \TeX Gyre, the latter based on a free counterpart of Adobe's basic set of 35 fonts and the former on Computer Modern. Both aim at greatly increasing the number of diacritical characters in the freely available collections of fonts.³²

Interesting for me with respect to Cyrillics.

Acknowledgements

Thank you NTG for inviting me for this conference.

Because of this invitation I contributed this paper, refreshed my \TeX knowledge, polished my pictures undusted `blue.tex`, and tried to get a modern communicating \TeX ↔MetaPost IDE running on my PC.³³

I hope you enjoyed reading this paper and if only one idea sparked up in your mind, I am happy. If the walk along history lane, enriched by my perspectives, did not interest you, hopefully the included pictures pleased you.

Thank you Jos Winnink for running the above `linear.i.mp` after 10+ years again, as well as the orphan `linear.i.mp`, which I also created in MetaFont in 1996, and just before this conference transcribed into MetaPost from memory, since I no longer had the original.³⁴

Thank you Taco Hoekwater for procrusting this note into the proceedings, and for your advice in getting me \TeX ing again.

The `linear.i` picture, I consider the most beautiful 2.5D graphics I ever made with \TeX , MetaFont, MetaPost, or PostScript, although `linear.i` comes close.

The doughnut picture was tricky.

I needed the equation solver of MetaFont, and thanks to my familiarity with splines I could reuse the MetaFont splines in PostScript. Recently, I made in MP a variant, see Appendix II.

The cat picture below is my oldest and my first exercise in MetaFont. I drew it already while at high school. Some years ago, I recast it into a wall sculpture composed of broken mirror pieces.

Cat
← drawing
sculpture →

I also made a puzzle of the cat drawing. Coloring the contours of the cat differently in each piece yielded a difficult, misleading puzzle: one has to concentrate on the drawing and not on the colors, nor the shape of the pieces.

Is this all? No, there is some more, but this is enough for the moment.

My case rests, have fun and all the best.



Notes

1. Both had courseware: NTG's Advanced T_EX course: Insight & Hindsight, respectively MetaFont: practical and impractical applications.
2. The L^AT_EX picture environment and the gkpmac suite I consider outdated, because of the inconsistency when using colors and because it is much better and more convenient to do

all your graphics in PS directly or via MetaPost.

3. At the conference I was reminded that BLUe is under copyright. I heard from the LPPL licensing and that seems enough for distributing it on the T_EX Live DVD. I agreed with Manfred Lotz to modify the copyright into LPPL, and to look over BLUe for release on T_EX Live DVD of 2010. As much as possible, for example the Publishing with T_EX guide, can already be released on the 2009 DVD as well as all my notes published in MAPS.

4. Kazimir Malevich, 1878–1935. Russian painter, born in Kiev.

5. At the conference Ulrik Vieth talked about the need for finetuning of math afterwards, especially with respect to a.o. adjusting spacing, aligning subscripts, and adapting for the size of delimiters. In an earlier paper OpenType Math Illuminated, BachoT_EX 2009, he details with OpenType Math compared to T_EX and MS Cambria release in Word 2007.

6. Hans Hagen reported about the Oriental T_EX project, which to me looks like an Oriental mode. Hans confirmed that one can look at it that way.

7. This is an old issue. In the past we had the expression American Screwdriver, meaning using your tool for everything. T_EX is not an American Screwdriver.

8. I did not say that one should work in plain or PS all the way. Of course one can start and write macros in whatever high level language is available. I do wish, when you are finished with the macros, that you translate them into plain, respectively PS, and include them in a library for reuse.

9. BachoT_EX2009.

10. Courtesy Phil Taylor, who published the macros for doing the calculation by using dimension variables.

11. \author is absent, because in BLUe the author is known, is default, you don't have to fill it in each time, similar holds for affiliation. BLUe is a personalized collection of macros.

12. When as a student I became member 1024 of the Nederlandse Rekenmachine Genootschap, I received The Art of Computer Programming I. My heroes next to Knuth are G. Ploya, G.E. Forsythe, Knuth's predecessor, C. Lanczos, F.L. Bauer, H. Rutishauser, P. Henrici, R.W. Hamming, L. Wall, and H.A. Lauwerier my Dutch applied Math professor. He would have loved my PS Impossible Cube, for sure.

13. Tony Hoare in the 70-ies coined the term in his famous paper.

14. SLC mean Slow, Steep, Strenuous Learning Curve.

15. Since T_EX was invented we have witnessed a tremendous development in computers, and how to use computers. The command line interface belongs to the past, we all use computers via GUIs. Why not have a Word-like document preparation system with T_EX as open, well-documented kernel, which can be accessed for advanced tasks?

16. Courtesy L. Wall.

17. Because L^AT_EX, ConT_EXt, BLUe and ilks have that, of course.

18. I did not say that one could start with the filename, because I consider that against what people are used to, and makes the problem a trifle. The specification was not watertight, because I preferred a more or less open problem, to stimulate creativity.

19. My MetaPost1.005 did not yet provide it.
20. This rotating shrinking squares and a few other pictures, which I borrowed from H.A. Lauwerier's 'Meetkunde met de microcomputer', such as the Koch fractal, the Hilbert curve, the Jura fractal, Escher's knot, . . . and published in MAPS in the mid-90-ies, in my 'Just a little bit of PostScript', 'Graphics & T_EX—a reappraisal of Metafont', or 'Tiling in PostScript and Metafont—Escher's wink', I found back, without reference and translated in MetaPost.
21. As yet not! Be aware that the PostScript code is just handed through, not taken notice of.
22. The numeric equation, $u=1\text{cm}$, looks a bit strange, looks more like a dimension à la T_EX. It becomes clear when you realize that cm is just a scaling factor.
23. The problem is that generally I got a picture per page, and I did not know how to trim the picture to its bounding box. After the conference I found out how to trim these pictures in Acrobat 7 professional: select the picture, copy it to the clipboard, and then click create PDF and select From Clipboard Image.
24. The conversion was generally done by opening the .pdf pictures in Photoshop, trim them and save them as .jpg. Later I became aware of the prologues:=3; statement, which yields a.o a picture trimmed to the Bounding Box.
25. M.C. Escher, 1898–1972, Dutch artist.
26. I must have the negative somewhere, but can't find it, alas. I'll give it another try.
27. Naum Gabo, 1890–1977. Born Naum Borisovich Pevsner. Bryansk. Russian Constructivist.
28. I colored the picture by post processing in Photoshop. A work flow approach is simpler via the use of the earlier defined \blue1 and switching back via \black.
29. The term hypertext was coined by TeD Nelson during the 1960s, the concept can be traced to Vanneger Bush in 1945. See the Hypertext issue of the CACM july, 1988.
30. I read in the PDF manual the excuse that PDF is a successor of PS, but does not have a programming language built in???
31. This wish will be fulfilled, as reported earlier.
32. Courtesy Bogusław Jackovski et al. and Ulrik Vieth.
33. Still a wish. But... Wybo installed Ubuntu Linux for me on my laptop, with T_EXworks, so I can explore that. Will give me enough work to do.
34. After the conference the NTG discussion list told me how to run the MetaPost utility supplied on T_EX Live. Open a new directory, copy cmd.exe into it as well as your filename.mp. Double click the cmd.exe icon and type after the prompt mpost filename.mp. Another suggestion was to use MetaPost from within the SciTE editor. It would have been handy if the readme, which comes with the T_EX Live, would have contained information on how to use the various utilities. Don't assume a casual user knows it.

Kees van der Laan
Hunzeweg 57, 9893PB Garnwerd, Groningen
kisa1@xs4all.nl

Appendix I: Contest Solutions

Phil Taylor submitted the following mean and lean post conference solution to the Contest, based on what is said on T_EXbook p204. He told me that the working of # at the end of the parameter list is little understood, but very useful.

Indeed, I had to look up that functionality and do the replacement to see what it does. A paradigm, though I don't know at the moment where Knuth used it. Phil's solution is a near solution, because curly braces are still needed around the filename.

But...

mean and lean it is, and I reused it already, adapted, for getting a list of all PS pictures used in this paper.

I realized, and use it as an aid in remembering, that the # at the end of the parameter list is a workaround for having a curly opening brace as separator.

```
\def \jpg #1#%
  {\def \next
    {\immediate \pdfximage #1
      {\the \toks 0 .jpg}
      \pdfrefximage \pdflastximage
    }
  \afterassignment \next
  \toks 0 =
  }
%Use
\jpg width 300pt height 30pt {P-Taylor}
```

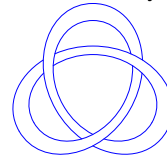
I came up with the solution below, which I will add to my FIFO paradigms list in my T_EXing Paradigms collection. Of course, I used this one for my purpose.

```
\def\jpgD#1\par{\def\scaling{ }
\ fifow#1 \wofif %Sentinels
\pdfximage\scaling\expandafter{\fn}
$$\pdfrefximage\pdflastximage$$}
%
\def\ fifow#1 #2{\process{#1}#2
\ ifx#2\wofif\expandafter\wofif\fi
\ fifow#2}
%
\def\wofif#1\wofif{ }
%
\def\process#1#2{%
\ ifx#2\wofif\def\fn{#1.jpg}%
\ else\edef\scaling{\scaling\space#1}%
\ fi}
```

Both solutions circumvent the pitfall of parsing the arguments. The height... and width... if present, are just passed through.

Appendix II: Escher's knot

The Escher's knot figure, I consider highly instructive. The latest version, in MP, makes use of the symmetry, while the hidden lines are **not** removed: the figure is (re)drawn with only the visible curve pieces. For constructing the pieces it is handy first to draw the complete picture wit dotlabel commands included, for the points P, Q, R, and the intersection points a, b, c, d. Construct with the aid of this picture the visible pieces anew from the calculated curves by the use of cutbefore.



```
u=5cm;
%Ext points P, Q, R, counter clockwise
pair P, dP; P:=(0,u); dP:=( 1, 0);
pair Q, dQ; Q:= P rotated 120;
pair R, dR; R:= P rotated 240;
dQ:=(-1, 1.73205);dR:=(-1, -1.73205);
path PQ, QR, RP, pq, qr, rp,
      PQa, PQb, pqa, pqb;%pieces
drawoptions(withcolor blue);
PQ = P{dP}.. .5R{dR}..{dQ}Q;
QR = PQ rotated 120;
RP = PQ rotated 240;
pq = PQ scaled .8;
qr = QR scaled .8;
rp = RP scaled .8;
%draw PQ..QR..RP;%No hidden lines removed
%draw pq..qr..rp;%No hidden lines removed
%Just the pieces instead of
%hidden lines removal
pqa = pq cutafter QR;
pqb = pq cutbefore qr;
draw pqa; draw pqb;
draw pqa rotated 120; draw pqb rotated 120;
draw pqa rotated 240; draw pqb rotated 240;
%a similar approach as above did not work, bug?
PQ:= PQ cutbefore QR;
PQa:= cuttings;
PQb:= PQ cutbefore qr;
draw PQa; draw PQb;
draw PQa rotated 120; draw PQb rotated 120;
draw PQa rotated 240; draw PQb rotated 240;
```

Note the names used: a path PQ is the 'line' between points P and Q. Very handy, this naming convention taken over from good old classical geometry. It facilitates reading of the code. With cutbefore and cutafter it seems more natural to draw just the visible pieces instead of erasing hidden parts.

LuaTeX lunatic

And Now for Something Completely Different
– Monty Python, 1972

Abstract

luatex lunatic is an extension of the Lua language of luatex to permit embedding of a Python interpreter.

A Python interpreter hosted in luatex allows macro programmers to use all modules from the Python standard library, allows importing of third modules, and permits the use of existing bindings of shared libraries or the creation of new bindings to shared libraries with the Python standard module ctypes.

Some examples of such bindings, particularly in the area of scientific graphics, are presented and discussed.

Intentionally the embedding of interpreter is limited to the python-2.6 release and to a luatex release for the Linux operating system (32 bit).

Keywords

Lua, Python, dynamic loading, ffi.

History

I met luatex sometime around November 2006, and I started to play with it to explore the possibility of typesetting xml documents in an alternative way than the traditional approach based on xsl stylesheet plus xslt processor.

My first intention was to typeset a wikipedia xml dump [4] which is compressed with bzip2; given that I mainly use Python for my programming language, I quickly found python-bz2 and then Gustavo Niemeyer’s “personal laboratory” [15] where I have discovered Lunatic Python [14].

To avoid confusion, here Python means CPython, the C implementation of the Python language [49]. There are other implementations of the Python language: for example Jython [41] and IronPython [37]. According to [6] “the origin of the name (is) based on the television series Monty Python’s Flying Circus.”

In March 2007 I started to investigate the possibility of integrating Lunatic Python with luatex [57] and in August 2007 I made the first release of luatex-lunatic [20], just around the birth of my second daughter Martina (09/08/07, [33]).

During the 2nd ConTeXt meeting [21] I found that luatex was stable enough to finalize the project, so I remade all steps and some new examples too (ConTeXt meetings are good places for these kinds of things).

Examples are now hosted at contextgarden [35] while [20] remains for historical purposes.

Motivations & goals

TeX is synonymous with portability (it’s easy to implement/adapt TeX *the program*) and stability (TeX *the language* changes only to fix errors).

We can summarize by saying that “*typesetting in TeX tends to be everywhere everytime.*”

These characteristics are a bit unusual in today’s scenario of software development: no one is surprised if programs exist only for one single OS (and even for a discontinued OS, given the virtualization technology) and especially no one is surprised at a new release of a program, which actually means bugs fixed and new features implemented (note that the converse is in some sense negative: no release means program discontinued).

Of course, if we consider the *L^ATeX-system*, i.e. L^ATeX and its most used packages, this is not frozen at all: just see the near-daily announcements from CTAN. pdfTeX also changes to follow pdf releases.

With luatex-lunatic I adopt this point of view: LuaTeX or more specifically LuaTeX & ConTeXt-mkiv as a **tool** for publishing content, with some extent to content management. As a tool, it is no surprise if there are “often” (for a TeX user) new releases, given that we can have a LuaTeX update, or a ConTeXt-mkiv update, or a Lua update, or a Python update, or a library update for which the Python binding exists; and, of course, if made with no “cum grano salis”, no surprise if this can become quickly unmanageable.

The price to pay is the potential **loss of stability**: the same document (with the same fonts and images) processed with a new release can produce a different output.

With regard to portability, the LuaTeX team uses libtool: *GNU Libtool simplifies the developer’s job by encapsulating both the platform-specific dependencies, and the user interface, in a single script. GNU Libtool is designed so that the complete functionality of each host type is available via a generic interface, but nasty quirks are hidden from the programmer [39]*, while in Lua and Python support for dynamic loading is a feature of the languages, i.e. there is a (Lua/Python) layer that hides the details of binding.

Thus stated, due to the lack of resources, I have no plan in the immediate future to investigate any OS other than Linux, so this article will cover this OS only; or, stated in another way, there is a potential **loss of portability**.

We can summarize saying that “*typesetting in luatex-lunatic is here and now*”, where *here* stands for “a specific OS” and *now* for “with this release”. Actually *here* stands for “Linux 32 bit”, and *now* stands for luatex-snapshot-0.42.0.tar.bz2 with ConTeXt-mkiv current 2008.07.17; probably both will already be old by the time this is printed.

Another motivation has emerged during the development of luatex-lunatic: the possibility to use ConTeXt-mkiv as a sort of literate programming tool for a specific context.

It is well known that CWEB is a way to tangle together a program written in a *specific* programming language (C) with its documentation written with a macro markup language, T_EX; luatex-lunatic and ConTeXt-mkiv can be used to tangle together a program written in an (almost) *arbitrary* programming language with its documentation written with a *high level* macro markup language, ConTeXt-mkiv.

Put in another way: currently an application calls T_EX or L^AT_EX (i.e. it creates a process) to obtain a result from a tex snippet (for example to show a math formula); instead luatex-lunatic with ConTeXt-mkiv calls the application by dynamic loading (i.e. it does not create a process) to obtain the result to insert into tex source.

For example one can use luatex-lunatic ConTeXt-mkiv to typeset a math formula, and the binding for the evaluation of the same formula (there are several symbolic-math Python modules already available).

We will see more about this later, when we will talk of Sage.

We want to find the smallest set of patches of the luatex codebase, or, better, we want to avoid:

1. constraints of any sort to the development team;
2. massive modifications of the code base;
3. radical modification of the building process.

Lunatic Python

There is no better site than [14] to explain what is Lunatic Python:

Lunatic Python is a two-way bridge between Python and Lua, allowing these languages to intercommunicate. Being two-way means that it allows Lua inside Python, Python inside Lua, Lua inside Python inside Lua, Python inside Lua inside Python, and so on.

...

The bridging mechanism consists of creating the missing interpreter state inside the host interpreter. That is, when you run the bridging system inside Python, a Lua interpreter is created; when you run the system inside Lua, a Python interpreter is created.

Once both interpreter states are available, these interpreters are provided with the necessary tools to interact freely with each other. The given tools offer not only the ability of executing statements inside the alien interpreter, but also to acquire individual objects and interact with them inside the native state. This magic is done by two special object types, which act by bridging native object access to the alien interpreter state.

Almost every object which is passed between Python and Lua is encapsulated in the language specific bridging object type. The only types which are not encapsulated are strings and numbers, which are converted to the native equivalent objects.

Besides that, the Lua side also has special treatment for encapsulated Python functions and methods. The most obvious way to implement calling of Python objects inside the Lua interpreter is to implement a `__call` function in the bridging object metatable. Unfortunately this mechanism is not supported in certain situations, since some places test if the object type is a function, which is not the case of the bridging object. To overwhelm these problems, Python functions and methods are automatically converted to native Lua function closures, becoming accessible in every Lua context. Callable object instances which are not functions nor methods, on the other hand, will still use the metatable mechanism. Luckily, they may also be converted in a native function closure using the `asfunc()` function, if necessary.



According to [68], page 47, a *closure* is “a function plus all it needs to access non-local variables correctly”; a non-local variable “is neither a global variable nor a local variable”. For example consider `newCounter`:

```
function newCounter()
  local i = 0
  return function()
    i = i+1
    return i
  end
end
c1 = newCounter()
print(c1()) --> 1
print(c1()) --> 2
c2 = newCounter()
print(c2()) --> 1
print(c1()) --> 3
print(c2()) --> 2
```

`i` is a non-local variable; we see that there is no interference between `c1` and `c2`—they are two different closures over the same function.

It's better to track a layout of installation of `luatex-lunatic` on a Linux box.

Let's set up a home directory:

```
HOMEDIR=/opt/luatex/luatex-lunatic
```

Next:

1. download and install `python-2.6.1` (at least) from [49]. Assuming `$HOMEDIR/Python-2.6.1` as build directory, let's configure `python-2.6.1` with

```
./configure
--prefix=/opt/luatex/luatex-lunatic
--enable-unicode=ucs4
--enable-shared
```

and install it. After installation we should end in a "Filesystem Hierarchy Standard"-like Filesystem (cf. [46], except for `Python-2.6.1`), i.e. something like this:

```
$> cd $HOMEDIR && ls -lX
bin
include
lib
man
share
Python-2.6.1
```

It's also convenient to extend the system path:

```
$> export PATH=
/opt/luatex/lunatic-python/bin:$PATH
```

so we will use the python interpreter in `$HOMEDIR`.

2. download `luatex` source code from [43]; we will use `luatex-snapshot-0.42.0`, so let's unpack it in `$HOMEDIR/luatex-snapshot-0.42.0`. For uniformity, make a symbolic link

```
$> cd $HOMEDIR
$> ln -s luatex-snapshot-0.42.0 luatex
```

It's convenient to have a stable `ConTeXt` minimals distribution installed (cf. [23]) under `$HOMEDIR`, i.e. `$HOMEDIR/minimals`, so that we will replace its `luatex` with our `luatex-lunatic`. Remember to set up the environment with

```
$> . $HOMEDIR/minimals/tex/setuptex
```

We don't build it now, because `build.sh` needs to be patched.

3. download `luatex-lunatic` from [3], revision 7, and put it in `lunatic-python`, i.e.

```
$> cd $HOMEDIR
$> bzip branch lp:lunatic-python
```

We must modify `setup.py` to match `luatex` installation (here "<" stands for the original `setup.py`, ">" stands for the modified one; it's a diff file):

```
1c1
< #!/usr/bin/python
---
> #!/opt/luatex/luatex-lunatic/bin/python
14,16c14,16
< LUALIBS = ["lua5.1"]
< LUALIBDIR = []
< LUAINCDDIR = glob.glob("/usr/include/lua*")
---
> LUALIBS = ["lua51"]
> LUALIBDIR = ['/opt/luatex/
luatex-lunatic/
luatex/build/tekk/web2c']
> LUAINCDDIR = glob.glob("../
luatex/source/tekk/web2c/luatexdir/lua51*")
48a49
>
```

When we build `lunatic-python`, we will end with a `python.so` shared object that will be installed in the `$HOMEDIR/lib/python2.6/site-packages` directory, so it's convenient to prepare a `python.lua` wrapper like this one:

```
loaded = false
func = package.loadlib(
"/opt/luatex/luatex-lunatic/lib/python2.6/
site-packages/python.so", "luaopen_python")
if func then
    func()
    return
end
if not loaded then
    error("unable to find python module")
end
```

Before building, we must resolve the dynamic loading problem; again from [14]

... Unlike Python, Lua has no default path to its modules. Thus, the default path of the real Lua module of Lunatic Python is together with the Python module, and a `python.lua` stub is provided. This stub must be placed in a path accessible by the `Lua require()` mechanism, and once imported it will locate the real module and load it.


Unfortunately, there's a minor inconvenience for our purposes regarding the Lua system which imports external shared objects. The hardcoded behavior of the

loadlib() function is to load shared objects without exporting their symbols. This is usually not a problem in the Lua world, but we're going a little beyond their usual requirements here. We're loading the Python interpreter as a shared object, and the Python interpreter may load its own external modules which are compiled as shared objects as well, and these will want to link back to the symbols in the Python interpreter. Luckily, fixing this problem is easier than explaining the problem. It's just a matter of replacing the flag RTLD_NOW in the loadlib.c file of the Lua distribution by the or'ed version RTLD_NOW|RTLD_GLOBAL. This will avoid "undefined symbol" errors which could eventually happen.

Modifying luatex/source/texk/web2c/
 luatexdir/lua51/loadlib.c
 is not difficult:

```
69c69
< void *lib = dlopen(path, RTLD_NOW);
---
> void *lib = dlopen(path, RTLD_NOW|RTLD_GLOBAL);
```

(again "<" means original and ">" means modified).

 According to dlopen(3) - Linux man page (see for example [18]),

The function dlopen() loads the dynamic library file named by the null-terminated string filename and returns an opaque "handle" for the dynamic library. If filename is NULL, then the returned handle is for the main program. If filename contains a slash ("/"), then it is interpreted as a (relative or absolute) pathname. Otherwise, the dynamic linker searches for the library as follows (see ld.so(8) for further details):

- (ELF only) If the executable file for the calling program contains a DT_RPATH tag, and does not contain a DT_RUNPATH tag, then the directories listed in the DT_RPATH tag are searched.
- If the environment variable LD_LIBRARY_PATH is defined to contain a colon-separated list of directories, then these are searched. (As a security measure this variable is ignored for set-user-ID and set-group-ID programs.)
- (ELF only) If the executable file for the calling program contains a DT_RUNPATH tag, then the directories listed in that tag are searched.
- The cache file /etc/ld.so.cache (maintained by ldconfig(8)) is checked to see whether it contains an entry for filename.
- The directories /lib and /usr/lib are searched (in that order).

If the library has dependencies on other shared libraries, then these are also automatically loaded by the dynamic linker using the same rules. (This process may occur recursively, if those libraries in turn have dependencies, and so on.)

One of the following two values must be included in flag:

- RTLD_LAZY
 Perform lazy binding. Only resolve symbols as the code that refer-

ences them is executed. If the symbol is never referenced, then it is never resolved. (Lazy binding is only performed for function references; references to variables are always immediately bound when the library is loaded.)

- RTLD_NOW
 If this value is specified, or the environment variable LD_BIND_NOW is set to a non-empty string, all undefined symbols in the library are resolved before dlopen() returns. If this cannot be done, an error is returned.

Zero or more of the following values may also be ORed in flag:

- RTLD_GLOBAL
 The symbols defined by this library will be made available for symbol resolution of subsequently loaded libraries.
- RTLD_LOCAL
 This is the converse of RTLD_GLOBAL, and the default if neither flag is specified. Symbols defined in this library are not made available to resolve references in subsequently loaded libraries.
- RTLD_NODELETE (since glibc 2.2)
 Do not unload the library during dlclose(). Consequently, the library's static variables are not reinitialised if the library is re-loaded with dlopen() at a later time. This flag is not specified in POSIX.1-2001.
- RTLD_NOLOAD (since glibc 2.2)
 Don't load the library. This can be used to test if the library is already resident (dlopen() returns NULL if it is not, or the library's handle if it is resident). This flag can also be used to promote the flags on a library that is already loaded. For example, a library that was previously loaded with RTLD_LOCAL can be re-opened with RTLD_NOLOAD | RTLD_GLOBAL. This flag is not specified in POSIX.1-2001.
- RTLD_DEEPBIND (since glibc 2.3.4)
 Place the lookup scope of the symbols in this library ahead of the global scope. This means that a self-contained library will use its own symbols in preference to global symbols with the same name contained in libraries that have already been loaded. This flag is not specified in POSIX.1-2001.

If filename is a NULL pointer, then the returned handle is for the main program. When given to dlsym(), this handle causes a search for a symbol in the main program, followed by all shared libraries loaded at program startup, and then all shared libraries loaded by dlopen() with the flag RTLD_GLOBAL.

External references in the library are resolved using the libraries in that library's dependency list and any other libraries previously opened with the RTLD_GLOBAL flag. If the executable was linked with the flag "-rdynamic" (or, synonymously, "-export-dynamic"), then the global symbols in the executable will also be used to resolve references in a dynamically loaded library.

If the same library is loaded again with dlopen(), the same file handle is returned. The dl library maintains reference counts for library handles, so a dynamic library is not deallocated until dlclose() has been called on it as many times as dlopen() has succeeded on it. The _init routine, if present, is only called once. But a subsequent call with RTLD_NOW may force symbol resolution for a library earlier loaded with RTLD_LAZY.

If dlopen() fails for any reason, it returns NULL. Nevertheless this is not enough: reference manual [66] says (page 23):

Dynamic loading of .so and .d11 files is disabled on all platforms.

So we must “enable” it and we must ensure that the `luatex` executable is linked against `libdl.so` because this contains the `dlopen()` symbol; also we must ensure that all the Lua functions involved in a `dlopen()` call must be resolved in the `luatex-lunatic` executable.

Assuming that we are always in `$HOMEDIR`, we must modify

```
source/teXk/web2c/luatexdir/am/liblua51.am
and source/teXk/web2c/Makefile.in.
```

For

```
source/teXk/web2c/luatexdir/am/liblua51.am:
```

```
12c12
< liblua51_a_CPPFLAGS += -DLUA_USE_POSIX
---
> liblua51_a_CPPFLAGS += -DLUA_USE_LINUX
```

while for `source/teXk/web2c/Makefile.in`:

```
98c98
< @MINGW32_FALSE@am__append_14 = -DLUA_USE_POSIX
---
> @MINGW32_FALSE@am__append_14 = -DLUA_USE_LINUX
1674c1674
< $(CXXLINK) $(luatex_OBJECTS) $(luatex_LDADD)
$(LIBS)
---
> $(CXXLINK) $(luatex_OBJECTS) $(luatex_LDADD)
$(LIBS) -Wl,-E -uluaL_openlibs -fvisibility=hidden
-fvisibility-inlines-hidden -ldl
```

The last patch is the most important, so let’s examine it more closely. Essentially, we are modifying the linking phase of building process of `luatex` (switch `-Wl,-E`) by adding `libdl` (switch `-ldl`) because `libdl` contains the symbol `dlopen` as stated before.

The switch `-uluaL_openlibs` tells the linker to consider the symbol `luaL_openlibs` even if it’s not necessary for building `luatex-lunatic`. In fact `luaL_openlibs` is coded in `lunatic-python/src/luainpython.c` and it needs to be resolved at runtime only when `luatex-lunatic` wants to load the Python interpreter.

So, even if `luaL_openlibs` is a function coded in `$HOMEDIR/luatex/source/teXk/web2c/luatexdir/lu51/limit.c`, it’s not used by `luatex`, so the linker discards this symbol because it’s useless.



According to `ld(1)`:

□ `-u` symbol

Force symbol to be entered in the output file as an undefined symbol. Doing this may, for example, trigger linking of additional modules from standard libraries. `-u` may be repeated with different option arguments to enter additional undefined symbols. This option is equivalent to the “`EXTERN`” linker script command.



It’s possible to examine how symbols are resolved runtime by setting `LD_DEBUG=all`; for example

```
$> export LD_DEBUG=all;
$> luatex python.lua &>python.LD_DEBUG;
$> export LD_DEBUG=
```

Here we are assuming a correct final `luatex lunatic luatex` and the `python.lua` wrapper seen before.

The file `python.LD_DEBUG` will show something like this:

```
3736: symbol=luaL_openlibs;
      lookup in file=./luatex-lunatic [0]
3736: binding file /opt/luatex/luatex-lunatic/
      lib/python2.6/site-packages/python.so [0]
      to ./luatex-lunatic [0]:
      normal symbol 'luaL_openlibs'
```


Without the `-uluaL_openlibs` linker flag, we will see something like this:

```
4033: symbol=luaL_openlibs;
      lookup in file=./luatex-lunatic-0.42.0.-test [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libm.so.6 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libdl.so.2 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/libreadline.so.5 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/libhistory.so.5 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/libncurses.so.5 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libc.so.6 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/ld-linux.so.2 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/opt/luatex/luatex-lunatic/lib/
      python2.6/site-packages/python.so [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libpthread.so.0 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libutil.so.1 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/tls/i686/cmox/libc.so.6 [0]
4033: symbol=luaL_openlibs;
      lookup in file=/lib/ld-linux.so.2 [0]
4033: /opt/luatex/luatex-lunatic/lib/python2.6/
      site-packages/python.so:
      error: symbol lookup error:
      undefined symbol: luaL_openlibs (fatal)
4033:
4033: file=/opt/luatex/luatex-lunatic/lib/python2.6/
      site-packages/python.so [0]; destroying link map
```

And near the bottom we can see this error: `symbol lookup error: undefined symbol: luaL_openlibs (fatal)`.

The last two switches, namely `-fvisibility=hidden` and `-fvisibility-inlines-hidden`, are `gcc` switches (not linker switches) and again they are related with

symbols, more precisely with symbols collisions. Consider this: in `$HOMEDIR/luatex/source/libs/libpng` there is a `libpng` library (currently vers. 1.2.38). This library, once compiled, will be merged by the linker into the `luatex` executable, and hence into the `luatex-lunatic` executable too. Now, we can build a Python binding to another `libpng` library or, better, we can import a Python module (e.g. `PythonMagickWand`, an interface to `ImageMagick`[®], see [40]) that has a binding to its own `libpng` library. In this situation, at runtime the dynamic loader will resolve for the Python module the symbols of `libpng` from `luatex libpng`, instead of those from its own `libpng`. Now, we cannot guarantee that these two libraries are the same, because we cannot replace the `libpng` of `luatex` (see near the end of the preceding section “Motivation & goals”) and, of course, we cannot replace the `libpng` library from the Python module with the one from `luatex`, because the last one can be patched for `luatex` only. So, we have symbols collisions (see [9]): almost for sure, a symbol collision will cause a segmentation fault, and the program abort.

 More information about this can be found starting from the already cited [9], especially [69]. A good text is also [63].

A solution can be this: “hide” to the “outside” all symbols that aren’t necessary for dynamic loading of shared objects. For standard `luatex`, this means “hide all”: for `luatex-lunatic`, this means “hide all but not symbols from lua”, otherwise we will not be able to use `loadlib`. It’s not so difficult to “hide all”: just patch the `build.sh` script of `luatex` sources by adding

```
28a29,36
> CFLAGS="-g -O2 -Wno-write-strings
    -fvisibility=hidden"
> CXXFLAGS="$CFLAGS
    -fvisibility-inlines-hidden"
> export CFLAGS
> export CXXFLAGS
```

The hardest part is to “unhide” the Lua part. We can proceed in this manner: collect the result of the patched `build.sh` in an out file:

```
$> cd $HOMEDIR/luatex; ./build.sh &> out
```

Then locate in `out` all the lines about Lua and remove the `-fvisibility=hidden` flag: for example

```
gcc -DHAVE_CONFIG_H -I.
-I../../../../source/teXk/web2c -I../..
-I/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/teXk
-I/opt/luatex/luatex-lunatic/
```

```
    luatex-snapshot-0.42.0/source/teXk
-I../../../../source/teXk/web2c/luatexdir/lu51
-DLUA_USE_LINUX -g -O2
-Wno-write-strings
-fvisibility=hidden
-Wdeclaration-after-statement
-MT liblua51_a-lapi.o
-MD -MP -MF .deps/liblua51_a-lapi.Tpo
-c -o liblua51_a-lapi.o
'test -f
'luatexdir/lu51/lapi.c'
|| echo
'../../../../source/teXk/web2c/'
    luatexdir/lu51/lapi.c
mv -f .deps/liblua51_a-lapi.Tpo
    .deps/liblua51_a-lapi.Po
```

will become

```
gcc -DHAVE_CONFIG_H -I.
-I../../../../source/teXk/web2c -I../..
-I/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/teXk
-I/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/source/teXk
-I../../../../source/teXk/web2c/luatexdir/lu51
-DLUA_USE_LINUX
-g -O2 -Wno-write-strings
-Wdeclaration-after-statement
-MT liblua51_a-lapi.o
-MD -MP -MF .deps/liblua51_a-lapi.Tpo
-c -o liblua51_a-lapi.o
'test -f
'luatexdir/lu51/lapi.c'
|| echo
'../../../../source/teXk/web2c/'
'luatexdir/lu51/lapi.c
mv -f .deps/liblua51_a-lapi.Tpo
    .deps/liblua51_a-lapi.Po
```

After that, recompile `luatex`

```
/bin/bash ./libtool
--tag=CXX
--mode=link
./CXXLD.sh -g -O2
-Wno-write-strings
-fvisibility=hidden
-fvisibility-inlines-hidden
-o luatex
luatex-luatex.o
libluatex.a libff.a
libluamisc.a libzzip.a
libluasocket.a liblua51.a
/opt/luatex/luatex-lunatic/
```

```

    luatex-snapshot-0.42.0/build/libs/
    libpng/libpng.a
/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/libs/
    zlib/libz.a
/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/libs/
    xpdf/libxpdf.a
/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/libs/
    obsdcompat/libopenbsd-compat.a
libmd5.a libmplib.a
lib/lib.a
/opt/luatex/luatex-lunatic/
    luatex-snapshot-0.42.0/build/tekk/
    kpathsea/libkpathsea.la
-lm -Wl,-E
-uluaL_openlibs
-fvisibility=hidden
-fvisibility-inlines=hidden
-ldl

```

Of course it's better to edit a `trick.sh` from out (see [34]) that will do all the work, paying the price of ~20 minutes of cut and paste for every new `luatex` release for preparing this trick file.

After executing `$HOMEDIR/luatex/trick.sh` we will have an *unstripped* `luatex` binary in `$HOMEDIR/luatex/build/tekk/web2c` so we are ready for the final step. It's better not to strip it, because we can track problems more easily.

- we copy `luatex` into the `bin` directory of `ConTeXt` minimals and remade formats:

```

$> cp $HOMEDIR/luatex/build/tekk/web2c/luatex
    $HOMEDIR/minimals/tekk/texmf-linux/bin
$> context --make

```

And in the end we must build the `lunatic-python` shared object:

```

$> cp $HOMEDIR/lunatic-python
$> python setup.py build && python setup.py
install

```



We can now make a simple test; let's save this in `test.tex`:

```

\directlua{require "python";
sys = python.import("sys");
tex.print(tostring(sys.version_info))}
\bye

```

Next let's run `callgrind`, a tool of `valgrind` (see [30]), to generate a *call graph* [5]:

```

$> valgrind --tool=callgrind
    --callgrind-out-file=test-%p.callgrind
    --dump-instr=yes
    luatex --fmt=plain --output-format=pdf test.tex

```

To see and analyze this call graph we can use `kcachegrind` [13]: see appendix at page 53 for the graph centered at `main` function, with `Min. node cost=1%`, `Min. call cost=1%`.

Examples

Image processing

ImageMagick. *ImageMagick* is “a software suite to create, edit, and compose bitmap images. It can read, convert and write images in a variety of formats (over 100) including DPX, EXR, GIF, JPEG, JPEG-2000, PDF, PhotoCD, PNG, PostScript, SVG, and TIFF. Use *ImageMagick* to translate, flip, mirror, rotate, scale, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bézier curves.” (See [40].) There are two bindings in Python, and we choose the `PythonMagickWand` [48], a `ctypes`-based wrapper for *ImageMagick*.



According to [50] `ctypes` is a foreign function library for Python. It provides C compatible data types, and allows calling functions in DLLs or shared libraries. It can be used to wrap these libraries in pure Python. `ctypes` is included in Python.

This simple script create a 200×200 pixel image at 300dpi with a shadow:

```

import PythonMagickWand as pmw
pmw.MagickWandGenesis()
wand = pmw.NewMagickWand()
background = pmw.NewPixelWand(0)
pmw.MagickNewImage(wand, 200, 200, background)
pmw.MagickSetImageResolution(wand, 118.110, 118.110)
pmw.MagickSetImageUnits(wand,
    pmw.PixelsPerCentimeterResolution)
pmw.MagickShadowImage(wand, 90, 3, 2, 2)
pmw.MagickWriteImage(wand, "out.png")

```

i.e., something like this:



Suppose we want to use it to generate a background for text, i.e.

```
\startShadowtext%
\input tufte
\stopShadowtext%
```

Let's now look at luatex lunatic and ConTeXt-mkiv in action for the first time:


```
\usetyescriptfile[type-gentium]
\usetyescript[gentium]
\setupbodyfont[gentium,10pt]
\setuppapersize[A6][A6]
\setuplayout[height=middle,topspace=1cm,
  header={2\lineheight},footer=0pt,backspace=1cm,
  margin=1cm,width=middle]
%%
%% lua layer
%%
\startluacode
function testimagemagick(box,t)
  local w
  local h
  local d
  local f
  local res = 118.11023622047244094488 -- 300 dpi
  local opacity = 25
  local sigma = 15
  local x = 10
  local y = 10
  w = math.floor((tex.wd[box]/65536 )
    /72.27*2.54*res)
  h = math.floor(((tex.ht[box]/65536)+
    (tex.dp[box]/65536))
    /72.27*2.54*res)
  f = string.format("%s.png", t)
  --
  -- Call the python interpreter
  --
  require("python")
  pmw = python.import("PythonMagickWand")
  wand = pmw.NewMagickWand()
  background = pmw.NewPixelWand(0)
  pmw.MagickNewImage(wand,w,h,background)
  pmw.MagickSetImageResolution(wand,res,res)
  pmw.MagickSetImageUnits(wand,
    pmw.PixelsPerCentimeterResolution)
  pmw.MagickShadowImage(wand,opacity,sigma,x,y)
  pmw.MagickWriteImage(wand ,f)
end
\stopluacode
%%
%% TeX layer
%%
```

```
\def\testimagemagick[#1]{%
\getparameters[Imgk][#1]%
\ctxlua{%
  testimagemagick(\csname Imgkbox\endcsname,
    "\csname Imgkfilename\endcsname")}%
}
%%
%% ConTeXt layer
%%
\newcount\shdw
\long\def\startShadowtext#1\stopShadowtext{%
\bgroup%
\setbox0=\vbox{#1}%
\testimagemagick[box=0,
  filename={shd-\the\shdw}]%
\defineoverlay[backg]%
  [{\externalfigure[shd-\the\shdw.png]}]%
\framed[background=backg,
  frame=off,offset=4pt]{\box0}%
\global\advance\shdw by 1%
\egroup%
}
\starttext
\startTEXpage%
\startShadowtext%
\input tufte
\stopShadowtext%
\stopTEXpage
\stoptext
```

As we can see, there is an almost one-to-one mapping between Python code and Lua code, a good thing for a small script.

And here is the result:

We thrive in information—thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeon-hole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsisize, winnow the wheat from the chaff and separate the sheep from the goats.

 What about symbols collisions?

```
$> eu-readelf --all luatex &> luatex.dump
$> export LD_DEBUG=all;context test-imagemagick.tex &> test-
  imagemagick.tex.LD_DEBUG; export LD_DEBUG=
```

If we search png_memcpy_check which is coded in \$HOMEDIR/source/libs/libpng/libpng-1.2.38/pngmem.c of luatex, we will find that

it's bound to system libpng;

```
25749: symbol=png_memcpy_check;
  lookup in file=luatex [0]
25749: symbol=png_memcpy_check;
  lookup in file=/lib/tls/i686/cmov/libm.so.6 [0]
25749: symbol=png_memcpy_check;
  lookup in file=/lib/tls/i686/cmov/libdl.so.2 [0]
:
: (62 lines after)
:
25749: symbol=png_memcpy_check;
  lookup in file=/usr/lib/libpng12.so.0 [0]
25749: binding file /usr/lib/libpng12.so.0 [0]
to /usr/lib/libpng12.so.0 [0]:
normal symbol 'png_memcpy_check' [PNG12_0]
```

In fact if we search for `png_memcpy_check` in `luatex.dump` we see that it's hidden now:

```
Symbol table [40] '.symtab' contains 10087 entries:
 9592 local symbols String table: [41] '.strtab'
:
Num:   Value   Size Type
4837: 082022b0  27 FUNC

Bind  Vis   Ndx Name
LOCAL  HIDDEN 13  png_memcpy_check
:
```

As a counterexample, suppose that we don't use hidden flags, so now `png_memcpy_check` is visible:

```
Num:   Value   Size Type
2273: 08243050  27 FUNC

Bind  Vis   Ndx Name
GLOBAL  DEFAULT 13  png_memcpy_check
```

Now we have a fatal error:

```
$> export LD_DEBUG=all;context test-imagemagick.tex &> test-
imagemagick.tex.LD_DEBUG; export LD_DEBUG=
:
MTXrun | fatal error, no return code, message: luatex: execu-
tion interrupted
:
```

and we see that `png_memcpy_check` is resolved in `luatex`:

```
24213: symbol=png_memcpy_check;
  lookup in file=luatex [0]
24213: binding file /usr/lib/libpng12.so.0 [0]
to luatex [0]:
normal symbol 'png_memcpy_check' [PNG12_0]
```

so we have symbols collisions. In this case it can be hard to track the guilty symbol; even in this case the fatal error can be given by another symbols collision, not necessarily `png_memcpy_check`. Also note that this code

```
\starttext
```

```
\externalfigure[out.png]
\stoptext
```

compiles right—of course, because there is no `PythonImageMagickWand` involved and so no symbols collisions. So this kind of error can become a nightmare.

Let's continue with our gallery.

PIL – PythonImageLibrary. PIL (see [51]) is similar to `ImageMagick`, but at least for `png` doesn't require `libpng`, so we are safe from symbol collisions.

```
\startluacode
function testPIL(imageorig,imagesepia)
  require("python")
  PIL_Image = python.import("PIL.Image")
  PIL_ImageOps = python.import("PIL.ImageOps")
  python.execute([[
def make_linear_ramp(white):
  ramp = []
  r, g, b = white
  for i in range(255):
    ramp.extend((r*i/255, g*i/255, b*i/255))
  return ramp
]])
  -- make sepia ramp
  -- (tweak color as necessary)
  sepia = python.eval
    ("make_linear_ramp((255, 240, 192))")
  im = PIL_Image.open(imageorig)
  -- convert to grayscale
  if not(im.mode == "L")
  then
    im = im.convert("L")
  end
  -- optional: apply contrast
  -- enhancement here, e.g.
  im = PIL_ImageOps.autocontrast(im)
  -- apply sepia palette
  im.putpalette(sepia)
  -- convert back to RGB
  -- so we can save it as JPEG
  -- (alternatively, save it in PNG or similar)
  im = im.convert("RGB")
  im.save(imagesepia)
end
\stoptluacode

\def\SepiaImage#1#2{%
\ctxlua{testPIL("#1", "#2")}%
\startcombination[1*2]
{\externalfigure[#1][width=512pt]}{\ss Orig.}
{\externalfigure[#2][width=512pt]}{\ss Sepia}
\stopcombination
}
```

```

\starttext
\startTEXpage
%\SepiaImage{lenna.jpg}{lenna-sepia.jpg}
\SepiaImage{lenna.png}{lenna-sepia.png}
\stopTEXpage
\stoptext

```

Here is the result (sorry, Lena is too nice to show her only in black and white):



The code shows how to define a Python function inside a Lua function and how to call it. Note that we must respect the Python indentation rules, so we can use the multiline string of Lua `[[...]]`.

Language adapter

Suppose we have a C library for a format of a file (i.e. TIFF, PostScript) that we want to manage in the same way as png, pdf, jpeg and jbig. One solution is to build a quick binding with ctypes of Python, and then import it

in `luatex-lunatic` as a traditional Lua module. As an example, let's consider `ghostscript` [10], here in vers. 8.64. It can be compiled as a shared library, and building a `testgs.py` (see [35]#Ghostscript) binding is not difficult (see file `base/gslib.c` in source distribution). The key here is to build a binding that fits our needs, not a general one.

```

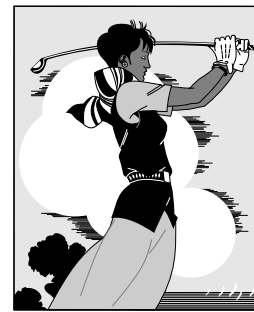
\startluacode
function testgs(epsin,pdfout)
  require("python")
  gsmodule = python.import("testgs")
  ghost = gsmodule.gs()
  ghost.appendargs('-q')
  ghost.appendargs('-dNOPAUSE')
  ghost.appendargs('-dEPSCrop')
  ghost.appendargs('-sDEVICE=pdfwrite')
  ghost.InFile = epsin
  ghost.OutFile = pdfout
  ghost.run()
end
\stopluacode

\def\epstopdf#1#2{\ctxlua{testgs("#1","#2")}}
\def\EPSfigure[#1]{%lazy way to load eps
\epstopdf{#1.eps}{#1.pdf}%
\externalfigure[#1.pdf]}

\starttext
\startTEXpage
{\EPSfigure[golfer]}
{\ss golfer.eps}
\stopTEXpage
\stoptext

```

Here is the result:



We can also use PostScript libraries: for example `barcode.ps` [56], a PostScript barcode library:

```

\startluacode
function epstopdf(epsin,pdfout)
  require("python")
  gsmodule = python.import("testgs")
  ghost = gsmodule.gs()
  ghost.appendargs('-q')

```

```

ghost.appendargs('-dNOPAUSE')
ghost.appendargs('-dEPSCrop')
ghost.appendargs('-sDEVICE=pdfwrite')
ghost.InFile = epsin
ghost.OutFile = pdfout
ghost.run()
end
function barcode(text,type,options,savefile)
require("python")
gsmodule = python.import("testgs")
barcode_string =
  string.format("%!\n100 100 moveto (%s) (%s)
%s barcode showpage",
                text,options,type)
psfile = string.format("%s.ps",savefile)
epsfile = string.format("%s.eps",savefile)
pdffile = string.format("%s.pdf",savefile)
temp = io.open(psfile,'w')
print(psfile)
temp:write(tostring(barcode_string),"\n")
temp:flush()
io.close(temp)
ghost = gsmodule.gs()
ghost.rawappendargs('-q')
ghost.rawappendargs('-dNOPAUSE')
ghost.rawappendargs('-sDEVICE=epswrite')
ghost.rawappendargs(
  string.format('-sOutputFile=%s',epsfile))
ghost.rawappendargs('barcode.ps')
ghost.InFile= psfile
ghost.run()
end
\stopluacode

\def\epstopdf#1#2{\ctxlua{epstopdf("#1","#2")}}
\def\EPSfigure[#1]{%lazy way to load eps
\epstopdf{#1.eps}{#1.pdf}%
\externalfigure[#1.pdf]%
}

\def\PutBarcode[#1]{%
\getparameters[bc][#1]%
\ctxlua{barcode("\csname bctext\endcsname",
                "\csname bctype\endcsname",
                "\csname bcoptions\endcsname",
                "\csname bcsavefile\endcsname" )}%
\expanded{\EPSfigure
          [\csname bcsavefile\endcsname]}%
}

\starttext
\startTEXpage
{\PutBarcode[text={CODE 39},type={code39},
options={includecheck includetext},
savefile={TEMP1}]}\\

```

```

{\ss code39}
\blank
{\PutBarcode[text={CONTEXT},type={code93},
options={includecheck includetext},
savefile={TEMP2}]}\\

{\ss code93}
\blank
{\PutBarcode[text={977147396801},type={ean13},
options={includetext},
savefile={TEMP3}]}\\

{\ss ean13}
\stopTEXpage
\stoptext

```

Of course one can implement a direct conversion into ps->pdf, instead of ps->eps->pdf. Here is the result:

For a beautiful book on PostScript see [58] (and its site [42]) and also [2].

Scientific & math extensions

Sage. “Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface. Mission: Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab.” [53]

Given that Sage comes with its own Python interpreter, we must rebuild lunatic-python and adapt python.lua accordingly; also sage is a command line program, so we need a stub `sagestub.py`:

```
from sage.all_cmdline import *
```

Here is the ConT_EXt-mkiv code:

```

\startluacode
function test_ode(graphout)
  require("python")
  pg = python.globals()
  SAGESTUB = python.import("sagestub")
  sage = SAGESTUB.sage
  python.execute([[
def f_1(t,y,params):
  return[y[1],
        -y[0]-params[0]*y[1]*(y[0]**2-1)]
]])
python.execute([[
def j_1(t,y,params):
  return [ [0,1.0],
           [-2.0*params[0]*y[0]*y[1]-1.0,
            -params[0]*(y[0]*y[0]-1.0)], [0,0]
]])

```

```

T=sage.gsl.ode.ode_solver()
T.algorithm="rk8pd"
f_1 = pg.f_1
j_1 = pg.j_1
pg.T=T
python.execute("T.function=f_1")
T.jacobian=j_1
python.execute("T.ode_solve(y_0=[1,0],
                    t_span=[0,100],
                    params=[10],num_points=1000)")
python.execute(string.format(
    "T.plot_solution(filename='%s')",
    graphout ))
end
\stopluacode

\def\TestODE#1{%
\ctxlua{test_ode("#1")}%
\startcombination[1*2]
{%
\ vbox{\hsize=8cm
Consider solving the Van der Pol oscillator
 $x''(t) + ux'(t)(x(t)^2-1)+x(t)=0$ 
between  $t=0$  and  $t= 100$ .
As a first order system it is
 $x'=y$ 
 $y'=-x+uy(1-x^2)$ 
Let us take  $u=10$  and use
initial conditions  $(x,y)=(1,0)$  and use the
\emphsl{\hbox{Runge-Kutta} \hbox{Prince-Dormand}}
algorithm.
}%
}{\ss \ }
{\externalfigure[#1][width=9cm]}{\ss Result
for 1000 points}}

\starttext
\startTEXpage
\TestODE{ode1.pdf}
\stopTEXpage
\stoptext

```

As we can see, here we use the `python.globals()` Lua function to communicate between the Python interpreter and Lua, and this can generate a bit of useless redundancy.

R. R “is a free software environment for statistical computing and graphics” [52]. It has its own language, but there is also a Python binding, `rpy2` [27], that we install in our `$HOMEDIR`.



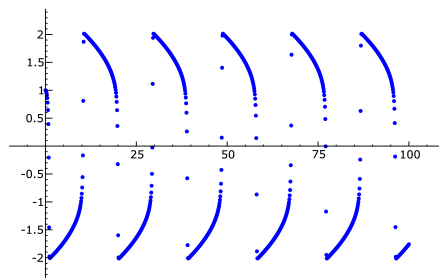
It can be necessary to add these env. variables

```

$>export R_HOME=/opt/luatex/luatex-lunatic/lib/R
$>export LD_PRELOAD=/opt/luatex/
          luatex-lunatic/lib/R/lib/libR.so

```

Consider solving the Van der Pol oscillator $x''(t) + ux'(t)(x(t)^2 - 1) + x(t) = 0$ between $t = 0$ and $t = 100$. As a first order system it is $x' = y$ $y' = -x + uy(1 - x^2)$ Let us take $u = 10$ and use initial conditions $(x,y) = (1,0)$ and use the *Runge-Kutta Prince-Dormand* algorithm.



Result for 1000 points

Figure 1. Result of the Sage code, with sage-3.2.3-pentiumM-ubuntu32bit-i686-Linux

For R we split the Python side in Lua in a pure Python script `test-R.py`:

```

import rpy2.robjects as robjects
import rpy2.rinterface as rinterface
class density(object):
    def __init__(self,samples,outpdf,w,h,kernel):
        self.samples = samples
        self.outpdf= outpdf
        self.kernel = kernel
        self.width=w
        self.height=h
    def run(self):
        r = robjects.r
        data = [int(k.strip())
                for k in
                file(self.samples,'r').readlines()
                ]
        x = robjects.IntVector(data)
        r.pdf(file=self.outpdf,
            width=self.width,
            height=self.height)
        z = r.density(x,kernel=self.kernel)
        r.plot(z[0],z[1],xlab='',ylab='')
        r['dev.off']()
if __name__ == '__main__' :
    dens =
    density('u-random-int','test-001.pdf',10,7,'o')
    dens.run()

```

and import this into Lua:

```
\startluacode
function testR(samples,output,w,h,kernel)
  require("python")
  pyR = python.import("test-R")
  dens =
    pyR.density(samples,output,w,h,kernel)
  dens.run()
end
\stopluacode

\def\plotdensiy[#1]{%
\getparameters[R][#1]%
\expanded{\ctxlua{testR("\Rsamples",
                        "\Routpdf",
                        "\Rwidth",
                        "\Rheight","\Rkernel")}}}}

\setupbodyfont[sans,14pt]
\starttext
\startTEXpage
\plotdensiy[samples={u-random-int},
             output={test-001.pdf},
             width={10},height={7},
             kernel={o}]

\setupcombinations[location=top]
\startcombination[1*2]
{\vbox{\hspace=400bp
This is a density plot of around {\tt 100 000}
random numbers between
 $\$0\$$  and  $\$2^{16}-1\$$  generated
from {\tt \hbox{/dev/urandom}}}}}{
{\externalfigure[test-001.pdf][width={400bp}]}
\stopcombination
\stopTEXpage
\stoptext
```

Note the conditional statement
`if __name__ == '__main__':`
 that permits to test the script with an ordinary Python
 interpreter.



It's worth noting that rpy2 is included in Sage too.

For more information about scientific computation with
 Python see [61] and [62] (also with site [31]) and [54].

The example of Sage shows that in this case we can
 think of `luatex lunatic` as an *extension* of Sage but
 also that `luatex lunatic` is *extended* with Sage. This
 is somehow similar to CWEB: code and description are
 tangled together, but now there's not a specific language
 like C in CWEB (in fact we can do the same with R).
 Eventually “untangled” is a matter of separation of

This is a density plot of around 100 000 random numbers between 0
 and $2^{16} - 1$ generated from `/dev/urandom`

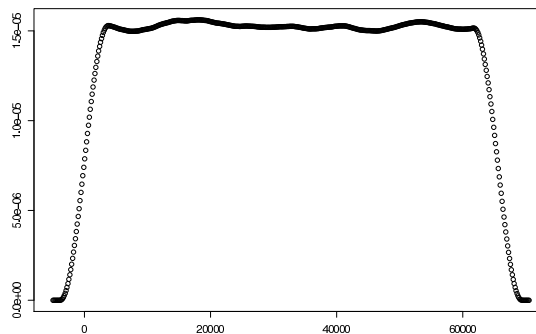


Figure 2. Result of the R code

Python code in a different file from the source tex file.



By the way, it's important to underline that CWEB is more
 advanced than other (C) code documentation systems because
 it embeds source code inside descriptive text rather than the reverse
 (as is common practice in most programming languages). Documenta-
 tion can be not “linear”, a bit unusual for ordinary programmers,
 but it's a more efficient and effective description of complex
 systems. Here we are talking about “linear” documentation,
 much like this article: a linear sequence of text-and-code
 printed as they appear.

Of course some computations may require much more
 time to be completed than the time of generation of the
 respective document (and `ConTeXt` is also a multipass
 system), so this approach is pretty inefficient—we need a
 set of macros that take care of intermediate results, i.e.
caching macros or *multipass macros*. For example, in
`ConTeXt-mkiv` we can say

```
\doifmode{*first}{%
  % this code will be executed only at first pass
  \Mymacro
}
```

so `\Mymacro` will be executed only at the first pass; there
 is also a Two Pass Data module `core-two.mkiv` that can
 be used for this, but don't forget that we also have Lua
 and Python for our needs.

Graphs

In `LuaTeX-ConTeXt-mkiv` we already have a very power-
 ful tool for technical drawing: `MetaPost`. Simple
 searching reveals for example `METAGRAPH` [32] or the
 more generic `LuaGRAPH` [19], a Lua binding to `graphviz`
 [38] with output also in `MetaPost`; both are capable of
 drawing (un)directed graphs and/or networks. The next
 two modules are more oriented to graph calculations.



MetaPost is an example of “embedding” an interpreter in LuaTeX at compilation time (see `luaopen_mplib(L)` in `void luainterpreter(void)` in `$HOMEDIR/source/teTeX/web2c/luatexdir/luauastuff.c`). So hosting Python is not a new idea: the difference is that the “embedding” is done at run time.

igraph. *igraph* “is a free software package for creating and manipulating undirected and directed graphs. It includes implementations for classic graph theory problems like minimum spanning trees and network flow, and also implements algorithms for some recent network analysis methods, like community structure search. The efficient implementation of *igraph* allows it to handle graphs with millions of vertices and edges. The rule of thumb is that if your graph fits into the physical memory then *igraph* can handle it. [11]



To install *igraph* we must first install *pycairo*, a Python binding to *cairo* [1], a well known 2D graphics library: so we gain another tool for generic drawing.

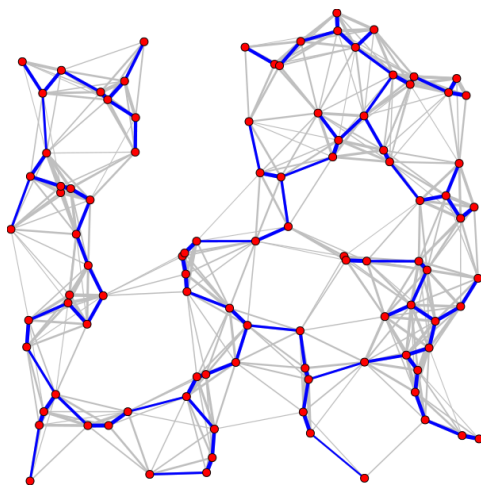


Figure 3. The result of the *igraph* code.

This time we coded the Python layer as a class:

```
import igragh
class spanningtree(object) :
    def __init__(self, ofn) :
        self.ofn = ofn
    def distance(self, p1, p2) :
        return ((p1[0]-p2[0]) ** 2
                + (p1[1]-p2[1]) ** 2) ** 0.5
    def plotimage(self) :
        res = igragh.Graph.GRG(100,
                                0.2, return_coordinates=True)
        g = res[0]
        xs = res[1]
        ys = res[2]
        layout = igragh.Layout(zip(xs, ys))
```

```
weights = [self.distance(layout[edge.source],
                          layout[edge.target]) for edge in g.es]
max_weight = max(weights)
g.es["width"] = \
[6 - 5*weight/max_weight for weight in weights]
mst = g.spanning_tree(weights)

fig = igragh.Plot(target=self.ofn)
fig.add(g, layout=layout,
        opacity=0.25,
        vertex_label=None)
fig.add(mst,
        layout=layout,
        edge_color="blue",
        vertex_label=None)
fig.save()
if __name__ == '__main__':
    sp = spanningtree('test-igraph.png')
    sp.plotimage()
```

In this case we calculate a minimum spanning tree of a graph, and save the result in `test-igraph.png`. The Lua layer is so simple that it is encapsulated in a TeX macro:

```
\def\PlotSpanTree#1{%
\startluacode
require("python")
local spantree_module
local sp
spantree_module =
python.import("test-igraph")
sp = spantree_module.spanningtree("#1")
sp.plotimage()
\stopluacode
\externalfigure[#1]}
\starttext
\startTeXpage
\PlotSpanTree{test-igraph.png}
\stopTeXpage
\stoptext
```

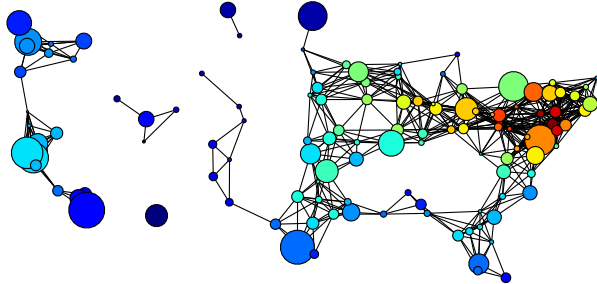
NetworkX. *NetworkX* is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. [22]

The code is simpler: we have only two layers: the Python layer, and the TeX layer. The Python layer is a trivial modification of `knuth_miles.py` (see [24], [36], [60]), and is left to the reader (hint: rename `..__init__..` in `def run()`).

```
\starttext
\startTeXpage
\ctxlua{require("python");
knuth=python.import("test-networkx");
knuth.run();}
```

```
\externalfigure[knuth_miles]
\stopTEXpage
\stoptext
```

Here is the result (with a bit of imagination, one can see the USA):



ROOT. *ROOT* is an object-oriented program and library developed by CERN. It was originally designed for particle physics data analysis and contains several features specific to this field. [7], [26]

In this example we will draw 110 lines from file `data` (each line being 24 float values separated by spaces); each line will be a curve to fit with a polynomial of degree 6. We isolate all relevant parts in a Python script `test-ROOT1.py`:

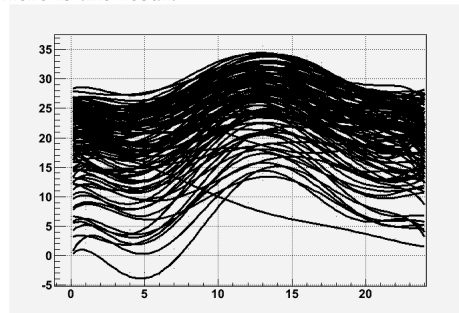
```
from ROOT import TCanvas,
    TGraph,TGraphErrors,TMultiGraph
from ROOT import gROOT
from math import sin
from array import array
def run(filename):
    c1 = TCanvas("c1", "multigraph", 200, 10, 700, 500)
    c1.SetGrid()
    mg = TMultiGraph()
    n = 24; x = array('d', range(24))
    data = file('data').readlines()
    for line in data:
        line = line.strip()
        y = array('d',
            [float(d) for d in line.split()])
        gr = TGraph(n,x,y)
        gr.Fit("pol6", "q")
        mg.Add(gr)
    mg.Draw("ap")
    c1.Update(); c1.Print(filename)
```

This is the Con \TeX t side:

```
\startluacode
function test_ROOT(filename)
    require("python")
    test = python.import('test-ROOT1')
    test.run(filename)
end
```

```
end
\stopluacode
\starttext \startTEXpage
\ctxlua{test_ROOT("data.pdf")}
\rotate[rotation=90]{\externalfigure[data.pdf]}
\stopTEXpage \stoptext
```

Here is the result:



Database

Oracle Berkeley DB XML. Oracle Berkeley DB XML “is an open source, embeddable xml database with XQuery-based access to documents stored in containers and indexed based on their content. Oracle Berkeley DB XML is built on top of Oracle Berkeley DB and inherits its rich features and attributes” [45];

We take as our data source a Wikiversity XML dump [8], more specifically `enwikiversity-20090627-pages-articles.xml`, a ~95MByte uncompressed xml file (in some sense, we end where we started).

Building a database is not trivial, so one can see [35] under `Build_the_container` for details. The most important things are indexes; here we use

```
container.addIndex("", "title",
    "edge-element-substring-string", uc)
container.addIndex("", "username",
    "edge-element-substring-string", uc)
container.addIndex("", "text",
    "edge-element-substring-string", uc)
```

These indexes will be used for substring queries, but not for regular expressions, for which it will be used the much slower standard way. Again it’s better to isolate the Python code in a specific module, `wikidbxml_queryTxn.py` (see [35] under `Make pdf` for details). This module does the most important work: translate from a ‘MediaWiki-format’ to Con \TeX t-mkiv. A ‘MediaWiki-format’ is basically made by `<page>` like this:

```
<page>
<title>Wikiversity:What is Wikiversity?</title>
```

```
<id>6</id>
<revision>
<id>445629</id>
<timestamp>2009-06-08T06:30:15Z</timestamp>
<contributor>
<username>Jr.duboc</username>
<id>138341</id>
</contributor>
<comment>/* Wikiversity for teaching */</comment>
<text xml:space="preserve">{{policy|[[WV:IS]]}}
{{about wikiversity}}
[[Image:Plato i sin akademi,
av Carl Johan Wahlbom
(ur Svenska Familj-Journalen).png
|thumb|left|300px|Collaboration between students
and teachers.]]
__TOC__
==Wikiversity is a learning community==
[[Wikiversity]] is a community effort to learn
and facilitate others'
learning. You can use Wikiversity to find
information or ask questions about a subject you
need to find out more about. You can also use it
to share your knowledge about a subject,
and to build learning
materials around that knowledge.
:
&lt;!-- That's all, folks! --&gt;
</text>
</revision>
</page>
```

So, a `<page>` is an xml document with non-xml markup in `<text>` node (which is an unfortunate tag name for an xml document); even if `<page>` is simple, parsing `<text>` content, or, more exactly, the text node of `<text>` node, is not trivial, and we can:

- implement a custom parser using the lpeg module of ConTeXt-mkiv (e.g. `$HOMEDIR/minimals/tex/texmf-context/tex/context/base/lxml-tab.lua`); this can be a good choice, because we can translate 'MediaWiki-format' directly into ConTeXt markup, but of course we must start from scratch;
- use an external tool, like the Python module `mwlib`: MediaWiki parser and utility library [25].

We choose `mwlib` (here in vers. 0.11.2) and implement the translation in two steps:

1. from 'MediaWiki-format' to XML-DocBook (more exactly DocBook RELAX NG grammar 4.4; see [44])
2. from XML-DocBook to ConTeXt-mkiv (this is done by the `getConTeXt(title, res)` function)



Actually, the `wikidbxml_queryTxn.writers()` function writes the result of the query by calling `wikidbxml_queryTxn.getArticleByTitle()` which in turn calls `wikidbxml_queryTxn.getConTeXt()` function.

The ConTeXt-mkiv side is (for the moment forget about the functions `listtitles(title)` and `simplereports(title)`):

```
\usetyescriptfile[type-gentium]
\usetyescript[gentium]
\setupbodyfont[gentium,10pt]
\setuppapersize[A5][A5]
\setuplayout[height=middle,
topspace=1cm,header={2\lineheight},
footer=0pt,backspace=1cm,margin=1cm,
width=middle]
%%
%% DB XML
%%
\startluacode
function testdbxml(title,preamble,
                    postamble,filename)
    require("python")
    pg = python.globals()
    wikiversity =
        python.import("wikidbxml_queryTxn")
    wikiversity.writers(title,preamble,
                        postamble,filename)
end
\stopluacode
%%
%% sqlite
%%
\startluacode
function listtitles(title)
    require("python")
    pg = python.globals()
    wikiversity =
        python.import("wikidbxml_queryTxn")
    r = wikiversity.querycategory(title)
    local j = 0
    local res = r[j] or {}
    while res do
        local d =
            string.format("%s\par",
                string.gsub(tostring(res),'_',' '))
        tex.sprint(tex.ctxcatcodes,d)
        j = j+1
        res = r[j]
    end
end
\stopluacode
%%
%% sqlite
```

```

%%
\startluacode
function simplereports(title)
  require("python")
  pg = python.globals()
  wikiversity =
    python.import("wikidbxml_queryTxn")
  r = wikiversity.simplereports(title)
  local j = tonumber(r)
  for v = 0,j-1 do
    local d =
      string.format("\input reps\%04d ",v)
    tex.sprint(tex.ctxcatcodes,d)
  end
  print( j )
end
\stopluacode
%% ConTeXt
\def\testdbxml[#1]{%
\getparameters[dbxml][#1]%
\ctxlua{%
testdbxml("\csname dbxmltitle\endcsname",
"\csname dbxmlpreamble\endcsname",
"\csname dbxmlpostamble\endcsname",
"\csname dbxmlfilename\endcsname")}%
\input \csname dbxmlfilename\endcsname %
}
\starttext
\testdbxml[title={Primary mathematics/Numbers},
preamble={},
postamble={},
filename={testres.tex}]
\stoptext

```

Here we query for the exact title Primary mathematics/Numbers: for the result, see page 55.

sqlite. Python offers adapters for practically all well known databases like PostgreSQL, MySQL, ZODB, etc. (“ZODB is a persistence system for Python objects” written in Python, see [16]. ZODB is the heart of Plone [47], a popular content management system also written in Python), but here we conclude with *sqlite*, a “*software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain*” (see [29]).

sqlite is a module of the standard Python library, and we can use it to query a `category.db` for titles. (`category.db` is a db made from `enwikiversity-20090627-category.sql`, which is a MySQL dump. Conversion is not difficult and is not shown here.)

The code uses the two functions seen before, `listtitles` and `simplereports`:

```

\starttext
{\bfb Query for 'geometr':}
\ctxlua{listtitles("geometr")}%
\ctxlua{simplereports("geometr")}%
\stoptext

```

See p. 57 for a short result (actually the first page of second hit, Geometry. The complete document is 12 pages).

MetaTeX

What is MetaTeX?

Quoting from `$HOMEDIR/tex/texmf-context/tex/context/base/metatex.tex`:

This format is just a minimal layer on top of the LuaTeX engine and will not provide high level functionality. It can be used as basis for dedicated (specialized) macro packages.

A format is generated with the command:

```
luatools --make --compile metatex
```

It should be clear from previous examples that a system with *all* these “bindings” becomes quickly unmanageable: one can spend almost all available time upgrading to latest releases. Just as an example: already at time of preprinting `ghostscript` was at rel. 8.70 (vs. rel. 8.64 of this article) Sage was at rel. 4.1 (vs. rel. 3.2.2), Python itself was at rel. 2.6.2 (vs. rel. 2.6.1) and there even exists rel. 3.1

Also not all Python packages are “robust”: for example, in the file `docbookwriter.py` of `mwlib` we can see

Generate DocBook from the DOM tree generated by the parser.

Currently this is just a proof of concept which is very incomplete

(and of course, `mwlib` was at rel. 0.12.2 (vs. rel. 0.11.2) so this message may have disappeared as well).

So, in the end, it’s better to have more distinct “tools” than one big tool that does anything and badly. We can see now why MetaTeX fits well in this line: it permits to create the exact “tool” needed and `luatex lunatic` can be used to complete this tool. For example, consider the problem of typesetting labels like the one on top if the next page.

Basically, it’s a table with barcode and two or three fonts (preferably monospaced fonts), most of the time black and white. `ConTeXt-mkiv` already comes with natural tables, or even a layer mechanism (see [70]); `luatex-lunatic` with `barcode.ps` provides the barcode. We don’t need colors, interaction, indexes, sectioning.



Financial reports are similar: here we can benefit from the decimal Python module that is included in the standard library (decimal is an implementation of Decimal fixed point and floating point arithmetic; see [28]).

MetaTeX can be used to produce very specific formats for educational purposes: think for example of a MetaTeX Sage, or a MetaTeX R from the CWEB point of view, i.e. embedded source code inside descriptive text rather than the reverse.

Also, Python can be used as a query language for Plone (mentioned previously), a powerful CMS written in Python, so it can be possible to print specific content type without translating it into an intermediate form like xml (and maybe in the future the reverse too, i.e. push a content type made by a MetaTeX Plone).

Conclusion

LuaTeX with ConTeXt-mkiv is a powerful tool for publishing content, and with an embedded Python interpreter we unquestionably gain more power, especially when MetaTeX becomes stabilized. If one wants, one can also experiment with JPyPe “*an effort to allow Python programs full access to Java class libraries. This is achieved not through re-implementing Python, as Jython/JPython has done, but rather through interfacing at the native level in both virtual machines*” [12] (currently unstable under Linux).

So it’s better here to emphasize “the dark side of the moon”.

First, it should be clear that currently we cannot assure **stability** and **portability** in the TeX sense.

Moreover, under Linux there is immediately a price to pay: symbol collisions. Even if the solution presented here should ensure that there are no symbol collisions between luatex and an external library, it doesn’t resolve problems of collision between symbols of two *external* libraries; installing all packages under a folder /opt/luatex/luatex-lunatic can help to track this problem, but it’s not a definitive solution. Of course, we avoid this problem if we use pure Python libraries, but these tend to be slower than C libraries.

ctypes looks fascinating, but a binding in ctypes is usually not easy to build; we must not forget that Lua offers its loadlib that can always be used as an alternative to ctypes or to any other Python alternative like SWIG [55] which can, anyway, build wrapper code for Lua too, at least from development release 1.3. In the end, an existing Python binding is a good choice if it is stable, rich, complete and mature with respect to an existing Lua binding, or if there is not a Lua binding.

For a small script, coding in Lua is not much different from coding in Python; but if we have complex objects, things can be more complicated: for example this Python code

```
z = x*np.exp(-x**2-y**2)
```

is translated in this not-so-beautiful Lua code

```
z=x.__mul__(np.exp((x.__pow__(2).
__add__(y.__pow__(2)))).__neg__())
```

(see [35]#Scipy). It is better to separate the Python layer into an external file, so we can eventually end in a `*py,*lua,*tex` for the same job, adding complexity to manage.

In the end, note that a Python interpreter does not “complete” in any sense luatex, because Lua is a perfect choice: it’s small, stable, and OS-aware. Conversely, Python is bigger, and today we are seeing Python versions 2.4, 2.5, 2.6.2, 2.7 alpha, 3.1 . . . not exactly a stable language from a TeX user point of view.

Acknowledgements

The author would like to thank Taco Hoekwater and Hans Hagen for their suggestions, help and encouragement during the development and the writing process of this article.

The author is also immensely grateful to Massimiliano “Max” Dominici for his strong support, help and encouragement.

References

All links were verified between 2009.08.17 and 2009.08.21.

- [1] <http://cairographics.org>
- [2] <http://cg.scs.carleton.ca/~luc/PSgeneral.html>
- [3] <https://code.launchpad.net/~niemeyer/lunatic-python/trunk>
- [4] <http://download.wikimedia.org>
- [5] http://en.wikipedia.org/wiki/Call_graphs
- [6] [http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://en.wikipedia.org/wiki/Python_(programming_language))
- [7] <http://en.wikipedia.org/wiki/ROOT>

- [8] http://en.wikiversity.org/wiki/Getting_stats_out_of_Wikiversity_XML_dumps
- [9] <http://gcc.gnu.org/wiki/Visibility>
- [10] <http://ghostscript.com/>
- [11] <http://igraph.sourceforge.net>
- [12] <http://jpype.sourceforge.net/>
- [13] <http://kcachegrind.sourceforge.net/cgi-bin/show.cgi>
- [14] <http://labix.org/lunatic-python>
- [15] <http://labix.org/python-bz2>
- [16] <https://launchpad.net/zodb>
- [17] <http://linux.die.net/man/1/dl>
- [18] <http://linux.die.net/man/3/dlopen>
- [19] <http://luagraph.luaforge.net/graph.html>
- [20] <http://luatex.bluwiki.com/go/User:Luigi.scarso>
- [21] <http://meeting.contextgarden.net/2008>
- [22] <http://networkx.lanl.gov>
- [23] <http://minimals.contextgarden.net/>
- [24] http://networkx.lanl.gov/examples/drawing/knuth_miles.html
- [25] <http://pypi.python.org/pypi/mwlib>
- [26] <http://root.cern.ch>
- [27] <http://rpy.sourceforge.net/>
- [28] <http://speleotrove.com/decimal>
- [29] <http://sqlite.org/>
- [30] <http://valgrind.org/>
- [31] <http://vefur.simula.no/intro-programming/>
- [32] <http://vigna.dsi.unimi.it/metagraph>
- [33] http://wiki.contextgarden.net/Future_ConTeXt_Users
- [34] <http://wiki.contextgarden.net/Image:Trick.zip>
- [35] http://wiki.contextgarden.net/User:Luigi.scarso/luatex_lunatic
- [36] <http://www-cs-faculty.stanford.edu/~knuth/sgb.html>
- [37] <http://www.codeplex.com/IronPython>
- [38] <http://www.graphviz.org>
- [39] <http://www.gnu.org/software/libtool>
- [40] <http://www.imagemagick.org/script/index.php>
- [41] <http://www.jython.org>
- [42] <http://www.math.ubc.ca/~cass/graphics/text/www/index.html>
- [43] <http://www.luatex.org>
- [44] <http://www.oasis-open.org/docbook>
- [45] <http://www.oracle.com/database/berkeley-db/xml/index.html>
- [46] <http://www.pathname.com/fhs/>
- [47] <http://www.plone.org>
- [48] <http://www.procoders.net/?p=39>
- [49] <http://www.python.org>
- [50] <http://www.python.org/doc/2.6.1/library/ctypes.html>
- [51] <http://www.pythonware.com/products/pil/>
- [52] <http://www.r-project.org/>
- [53] <http://www.sagemath.org/>
- [54] <http://www.scipy.org/>
- [55] <http://www.swig.org>
- [56] <http://www.terryburton.co.uk/barcodewriter/>
- [57] private email with Taco Hoekwater
- [58] Bill Casselman, *Mathematical Illustrations: A Manual of Geometry and PostScript*. ISBN-10: 0521547881, ISBN-13: 9780521547888 Available at site <http://www.math.ubc.ca/~cass/graphics/text/www/index.html>
- [59] Danny Brian, *The Definitive Guide to Berkeley DB XML*. Apress, 2006. ISBN-13: 978-1-59059-666-1
- [60] Donald E. Knuth, *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM Press, New York, 1993. ISBN 978-0-470-75805-2
- [61] Hans Petter Langtangen, *A Primer on Scientific Programming with Python*. Springer, 2009. ISBN: 978-3-642-02474-0
- [62] Hans Petter Langtangen, *Python Scripting for Computational Science*. Springer, 2009. ISBN: 978-3-540-73915-9
- [63] John Levine, *Linkers & Loaders*. Morgan Kaufmann Publisher, 2000. ISBN-13: 978-1-55860-496-4
- [64] Mark Lutz, *Learning Python, Fourth Edition*. O'Reilly, September 2009 (est.) ISBN-10: 0-596-15806-8, ISBN 13: 978-0-596-15806-4
- [65] Mark Lutz, *Programming Python, Third Edition*. O'Reilly, August 2006. ISBN-10: 0-596-00925-9, ISBN 13: 978-596-00925-0
- [66] [luatexref-t.pdf](#). Available in manual folder of [luatex-snapshot-0.42.0.tar.bz2](#)
- [67] Priscilla Walmsley, *XQuery*. O'Reilly, April 2007. ISBN-13: 978-0-596-00634-1
- [68] Roberto Ierusalimschy, *Programming in Lua (second edition)*. Lua.org, March 2006. ISBN 85-903798-2-5
- [69] Ulrich Drepper, *How to Write Shared Libraries*. <http://people.redhat.com/drepper/dsohowto.pdf>
- [70] Willi Egger, ConTeXt: Positioning design elements at specific places on a page (tutorial). EuroT_EX 2009 & 3rd ConT_EXt Meeting
- [71] Yosef Cohen and Jeremiah Cohen, *Statistic and Data with R*. Wiley 2008. ISBN 978-0-470-75805-2

I currently use Ubuntu Linux, on a standalone laptop—it has no Internet connection. I occasionally carry flash memory drives between this machine and the Macs that I use for network surfing and graphics; but I trust my family jewels only to Linux.

— Donald Knuth

Interview with Donald Knuth

By Donald E. Knuth and Andrew Binstock

Apr. 25, 2008

<http://www.informit.com/articles/article.aspx?p=1193856>

The lunatic is on the grass

The lunatic is on the grass

Remembering games and daisy chains and laughs

Got to keep the loonies on the path

— Brain Damage,

The Dark Side of the Moon,

Pink Floyd 1970

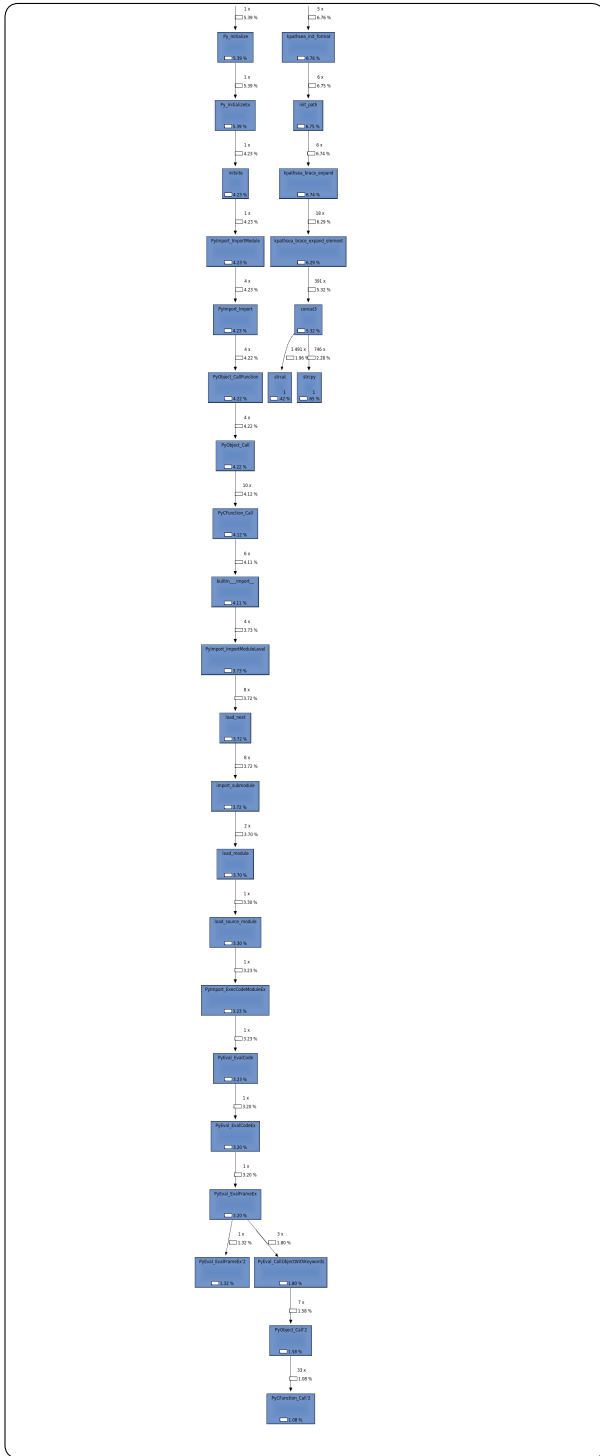
Mr. LuaT_EX hosts a Python,

and become a bit lunatic

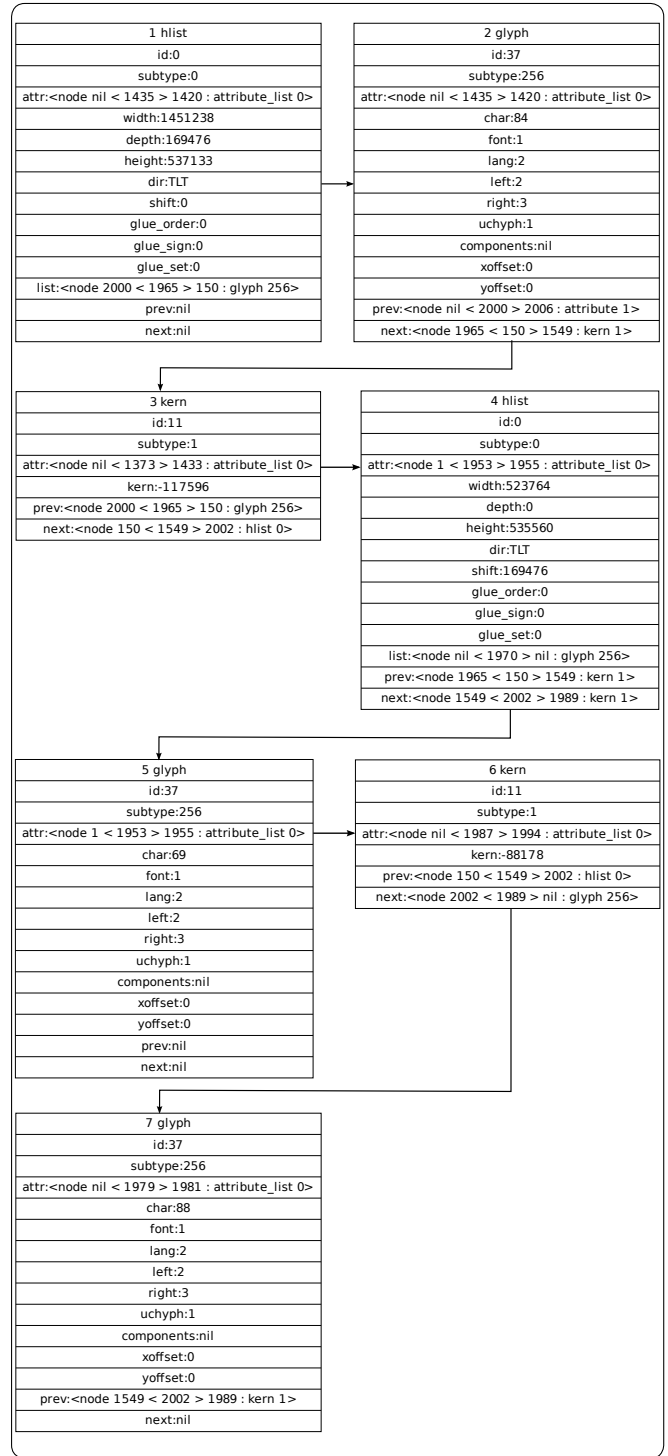
— Anonymous

Luigi Scarso

Call graph of a simple run, cont.



T_EX, forever



T_EX nodelist made with lunatic binding for graphviz

DB XML example

1 Primary mathematics/Numbers

1.1 Primary mathematics/Numbers

1.1.1 Teaching Number

This page is for teachers or home-schoolers. It is about teaching the basic concepts and conventions of simple number.

1.1.1.1 Developing a sound concept of number

Children typically learn about numbers at a very young age by learning the sequence of words, "one, two, three, four, five" etc. Usually, in chanting this in conjunction with pointing at a set of toys, or mounting a flight of steps for example. Typically, 'mistakes' are made. Toys or steps are missed or counted twice, or a mistake is made in the chanted sequence. Very often, from these sorts of activities, and from informal matching activities, a child's concept of number and counting emerges as their mistakes are corrected. However, here, at the very foundation of numerical concepts, children are often left to 'put it all together' themselves, and some start off on a shaky foundation. Number concepts can be deliberately developed by suitable activities. The first one of these is object matching.

1.1.2 Matching Activities

As opposed to the typical counting activity children are first exposed to, matching sets of objects gives a firm foundation for the concept of number and numerical relationships. It is very important that matching should be a physical activity that children can relate to and build on.

Typical activities would be a toy's tea-party. With a set of (say) four toy characters, each toy has a place to sit. Each toy has a cup, maybe a saucer, a plate etc. Without even mentioning 'four', we can talk with the child about 'the right number' of cups, of plates etc. We can talk about 'too many' or 'not enough'. Here, we are talking about number and important number relations without even mentioning which number we are talking about! Only after a lot of activities of this type should we talk about specific numbers and the idea of number in the abstract.

1.1.3 Number and Numerals

Teachers should print these numbers or show the children these numbers. Ideally, the numbers should be handled by the student. There are a number of ways to achieve this: cut out numerals from heavy cardstock, shape them with clay together, purchase wooden numerals or give them sandpaper numerals to trace. Simultaneously, show the definitions of these numbers as containers or discrete quantities (using boxes and small balls, eg. 1 ball, 2 balls, etc. Note that 0 means "no balls"). This should take some time to learn thoroughly (depending on the student).

0 1 2 3 4 5 6 7 8 9

1.1.4 Place Value

The Next step is to learn the place value of numbers.

It is probably true that if you are reading this page you know that after 9 comes 10 (and you usually call it ten) but this would not be true if you would belong to another culture.

Take for example the Maya Culture where there are not the ten symbols above but twenty symbols.

cfr <http://www.michielb.nl/maya/math.html>

Imagine that instead of using 10 symbols one uses only 2 symbols. For example 0 and 1

Here is how the system will be created:

Binary	0	1	10	11	100	101	110	111	1000	...
Decimal	0	1	2	3	4	5	6	7	8	...

Or if one uses the symbols A and B one gets:

Binary	A	B	BA	BB	BAA	BAB	BBA	BBB	BAAA	...
Decimal	0	1	2	3	4	5	6	7	8	...

This may give you enough information to figure the place value idea of any number system.

For example what if you used 3 symbols instead of 2 (say 0,1,2).

Trinary	0	1	2	10	11	12	20	21	22	100	...
Decimal	0	1	2	3	4	5	6	7	8	9	...

If you're into computers, the HEXADECIMAL (Base 16) or Hex for short, number system will be of interest to you. This system uses 4 binary digits at a time to represent numbers from 0 to 15 (decimal). This allows for a more convenient way to express numbers the way computers think - that we can understand. So now we need 16 symbols instead of 2, 3, or 10. So we use 0123456789ABCDEF.

Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000	...
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	...

1.1.5 Resources for Early Math

15 Fun Ways and Easy Games for Young Learners Math: Reproducible, Easy-to-Play Learning Games That Help Kids Build Essential Math Skills, by Susan Julio, Scholastic Professional, 2001.

Enie Meenie Miney Math!: Math Play for You and Your Preschooler, by Linda Allison, Little Brown & Co., 1993.

Marshmallow Math: Early Math for Toddlers, Preschoolers and Primary School Children, by Trevor Schindeler, Trafford, 2002.

Number Wonder: Teaching Basic Math Concepts to Preschoolers, by Deborah Saathoff and Jane Jarrell, Holman Bible, 1999.

cfr Category:School of Mathematics

cfr Category:Pages moved from Wikibooks

cfr Category:Primary education

Next in Primary School Mathematics:

cfr http://en.wikiversity.org/wiki/Primary_mathematics:Adding_numbers

sqlite example

1

Query for 'geometr': Geometric algebra

Geometry

Introductory Algebra and Geometry

Orbital geometry

Coordinate Geometry

2 Geometry

2.1 Geometry

This subdivision is dedicated to bridging the gap between the mathematical layperson and the student who is ready to learn calculus and higher mathematics or to take on any other endeavour that requires an understanding of basic algebra and (at least Euclidean) geometry.

2.1.1 Subdivision news

2.1.2 Departments

2.1.3 Active participants

The histories of Wikiversity pages indicate who the active participants are. If you are an active participant in this subdivision, you can list your name here (this can help small subdivisions grow and the participants communicate better; for large subdivisions a list of active participants is not needed). Please remember: if you have an

cfr <http://en.wikiversity.org/w/index.php?title=Special:Userlogin&type=signup>

cfr Category:Geometry

cfr Category:Introduction

cfr \#

cfr \#

In Cartesian or Analytic Geometry we will learn how to represent points, lines, and planes using the Cartesian Coordinate System, also called Rectangular Coordinate System. This can be applied to solve a broad range of problems from geometry to algebra and it will be very useful later on Calculus.

2.1.4 Cartesian Coordinate System

The foundations of Analytic Geometry lie in the search for describing geometric shapes by using algebraic equations. One of the most important mathematicians that helped to accomplish this task was René Descartes for whom the name is given to this exciting subject of mathematics.

2.1.4.1 The Coordinate System

For a coordinate system to be useful we want to give to each point an attribute that help to distinguish and relate different points. In the Cartesian system we do that by describing a point using the intersection of two(2D Coordinates) or more(Higher Dimensional Coordinates) lines. Therefore a point is represented as $P(x_1, x_2, x_3, \dots, x_n)$ in "n" dimensions.

2.1.5 Licensing:

"Geometry is the only science that it hath pleased God hitherto to bestow on mankind."—Thomas Hobbes

This department of the Olympiad Mathematics course focuses on problem-solving based on circles and vectors, thus generalizing to Coordinate Geometry. Our major focus is on Rectangular (Cartesian) Coordinates, although the course does touch upon Polar coordinates.

The first section is based on the geometric study of circles. Although not based on pure analytical geometry, it uses Apollonius-style reference lines in addition to Theorems on Tangents, Areas, etc.

The second section is devoted to Vector Analysis, covering problem-solving from Lattices and Affine Geometry to Linear Algebra of Vectors

Third section, focusing on locus problems, is all about conic sections and other curves in the Cartesian plane.

2.1.6 Textbooks

2.1.7 Practice Questions

Decorating CD-ROMs and DVDs (Tutorial)

Abstract

After having burned a disk you sometimes need to add a label and, if the disk is stored in a jewel case, a booklet and an inlay for the jewel case. The following article describes how to create a label for the disk on a commercial label-sheet and a booklet and an inlay for the jewel case. The following solutions are based on ConTeXt's built-in layer capabilities.

Keywords

ConTeXt, CD-ROM, DVD, label, booklet, inlay, layer.

Label

The label's several elements can be switched on/off with the following three modes:

- enabling *draft* mode will draw an outline of the label. Beware, simultaneously enabling the *withBackground* mode will obscure the outline.
- enabling *withLogo* mode will show an image at the top of the label.
- enabling *withBackground* mode will place the background image on the label.

```
\enablemode[draft]
\enablemode[withLogo]
\enablemode[withBackground]
```

We begin by specifying the main language.

```
\mainlanguage[en]
```

Next, we choose the label font. Since we are using ConTeXt MkIV, we will choose the Iwona-medium font in its *off* variant.

```
\usetypescript[iwona-medium]
\setupbodyfont[iwona-medium, 10pt]
```

All texts in the different boxes on the various layers are of type `\framed`. As a visual aid, *draft* mode switches on all their frames.

```
\doifmodeelse{draft}
  {\def\Onoff{on}}
  {\def\Onoff{off}}
```

We will place two images in the background. As a convenience, we define two macros whose names are the names of their respective images. The *Labelbackground* image covers the complete label, while the *Logo* image will appear at the top of the label when the *withLogo* mode is enabled.

```
\def\Labelbackground{fish}
\def\Logo{eurotexlogo}
```

Texts can be placed at the top and bottom of the label, and to the left and right of the disk's center. The bottom area is divided into three sections, each wide enough to fit, depending on the position of the text, within the available space.

To keep things uncluttered, we place the texts in buffers here and use only the buffers' names in the later typesetting instructions. All the texts will be placed in the `\framed` environment.

```
\startbuffer[BottomtextA]
  \green Contains all files for decorating a CD or DVD:\crlf
  \em \tfx CD-label, CD-inlay-booklet, CD-inlay for the jewel case
\stopbuffer

\startbuffer[BottomtextB]
  \green \em Published in the Euro\TEX -proceedings
\stopbuffer

\startbuffer[BottomtextC]
  \green 2009
\stopbuffer
```

There is one text area at the top of the label.

```
\startbuffer[Toptext]
  {\tfc \red CD/DVD decoration}\blank \green{\tfb Tutorial}\vfill Over-
lays and layers
\stopbuffer
```

The text areas to the left and right of the disk's center are, like the area at the top of the label, based on a single block of text.

```
\startbuffer[Righttext]
  \green CD-ROM
  \blank
  \gray \currentdate
\stopbuffer

\startbuffer[Lefttext]
  \green \TEX -tutorial
  \blank
  Euro\TEX 2009
\stopbuffer
```

By separating the content elements above from the typesetting commands below, we can change the content without worrying about the code below this point. We will add a comment to emphasize this.

```
% -- Basically you do not need to edit the lines below
```

First, we tell ConT_EXt how the label will look when typeset.

```
\setuppapersize[A4][A4]
\setuppagenumbering[state=stop]
\setupcolors[state=start]

\setuplayout
  [topspace=0pt,
  backspace=0pt,
  header=0pt,
  footer=0pt,
  margin=0pt,
  width=210mm,
  height=297mm,
  marking=on,
  location=middle]
```

As mentioned above, enabling *draft* mode will draw the label outline. The drawing itself consists of two concentric circles drawn with MetaPost.

```
\startreusableMPgraphic{CDShape}
  draw fullcircle scaled 117mm;
  draw fullcircle scaled 41mm;
\stopreusableMPgraphic
```

ConT_EXt provides the `\doifmode[]{}{}` command, which we will use to set the label background to our predefined background image when the *withBackground* mode is enabled.

```
\doifmode{withBackground}
  {\defineoverlay
   [Lbackground]
   [{\externalfigure
    [\Labelbackground]
    [width=\overlaywidth,height=\overlayheight]}}}
```

We use the same mechanism to place the optional logo image.

```
\doifmode{withLogo}
  {\defineoverlay
   [Logo]
   [{\externalfigure[\Logo]
    [width=\overlaywidth,height=\overlayheight]}}}
```

Here we define a layer that will cover the entire page. To indicate that we do not want relative positioning of this layer, we set its position to `no` and its reference point to `bottom`. Lastly, we place the layer into the page area as a background.

```
\definelaye[PageLayer][position=no]
\setuplayer
 [PageLayer]
 [corner=bottom,location=c,height=\paperheight]
\setupbackgrounds[page][background=PageLayer]
```

We define a second layer to hold the label fields we have already defined. We set this layer's reference point to `top` and `left` and the location of the layer data to `bottom right`. We also define its width and height. Set `option=test` to see what ConT_EXt does with these settings.

```
\definelaye
 [Label]
 [position=no,corner={top,left},location={bottom,right},
  width=117mm,height=117mm,option=test]
```

In the following lines we fill the Label layer with our predefined label fields, and then typeset it. To tell ConT_EXt to flush the layer at the end of the page we enclose the filling commands and the typesetting command: `\placelayer[]` within a `\standardmakeup ... \stopstandardmakeup` block.

```
\starttext
\startstandardmakeup[page=no,doublesided=no]
\setlayerframed
 [Label]
 [x=\dimexpr(\textwidth-117mm)/2,y=21.43mm]
 [width=117mm,height=117mm,frame=\Onoff,background=Lbackground]
 {}
```

```

\setlayerframed
[Label]
[x=\dimexpr(\textwidth-13mm)/2,y=22.43mm]
[width=13mm,height=13mm,frame=\Onoff,align={top,middle},
background=Logo]
{}
\setlayerframed
[Label]
[x=\dimexpr(\textwidth-78mm)/2,y=35.43mm]
[width=78mm,height=24mm,frame=\Onoff,align={top,middle}]
{\getbuffer[Toptext]}
\setlayerframed
[Label]
[x=126mm,y=60mm]
[width=34mm,height=40mm,frame=\Onoff,align={middle,lohi}]
{\getbuffer[Righttext]}
\setlayerframed
[Label]
[x=50mm,y=60mm]
[width=34mm,height=40mm,frame=\Onoff,align={flushleft,lohi}]
{\getbuffer[Lefttext]}
\setlayerframed
[Label]
[x=\dimexpr(\textwidth-98mm)/2,y=100.43mm]
[width=98mm,height=\dimexpr(38mm/3),frame=\Onoff,align=middle]
{\getbuffer[BottomtextA]}
\setlayerframed
[Label]
[x=\dimexpr(\textwidth-72mm)/2,y=\dimexpr(100.43mm+38mm/3)]
[width=72mm,height=\dimexpr(38mm/3),frame=\Onoff,align=middle]
{\getbuffer[BottomtextB]}
\setlayerframed
[Label]
[x=\dimexpr(\textwidth-18mm)/2,y=\dimexpr(100.43mm+38mm/3*2)]
[width=18mm,height=\dimexpr(38mm/3),frame=\Onoff,align=middle]
{\getbuffer[BottomtextC]}

\doifmode{draft}
{\setlayer[PageLayer][x=.5\paperwidth,y=216.93mm]
{\useMPgraphic{CDShape}}
\setlayer[PageLayer][x=.5\paperwidth,y=79.93mm]
{\useMPgraphic{CDShape}}}
\placelayer[Label]
\stopstandardmakeup
\stoptext

```

As you can see in the code above, we move a piece of information to its correct position by adjusting its vertical and horizontal offsets.

Near the end of the code there is another conditional action that controls the placement of the label shape.

The preceding code yields the following result:

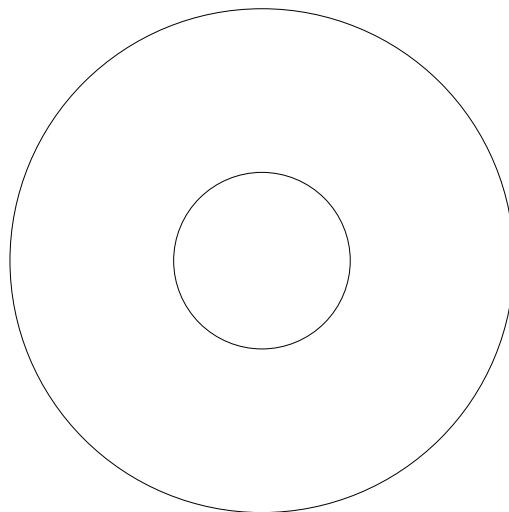
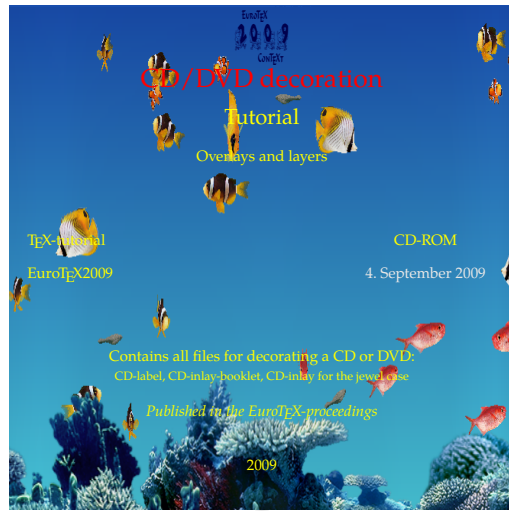


Figure 1. Example CD-label

Booklet

The CD-booklet is composed of a single section which we arrange with `\setuparranging[2UP]`. As before, we start with setting the main language and choosing the font for the booklet.

```
\mainlanguage[en]
\usetyescript[palatino]
\setupbodyfont[palatino,8pt]
```

If you want to place a background image on the front page, you can define a macro with a symbolic name which will be used later when you setup the page layer.

```
\def\Pageimage{fish}
```

We can add a title typeset along the spine of the cover page. The title is placed in a buffer:

```
\startbuffer[Sidetitle]
  {\tfc \yellow CD/DVD decoration}
\stopbuffer
```

In the next buffer we place the contents of the cover page.

```
\startbuffer[Covertext]
  \strut
  \blank[line]
  \startalignment[middle]
  \startlines
    {\tfc \red CD/DVD decoration}
    \blank \green{\tfb Tutorial}
    \blank Overlays and layers
  \stoplines
  \stopalignment
  \blank
\stopbuffer
```

The booklet contents is put into its own buffer.

```
\startbuffer[Bookletcontent]
  \input knuth\par
\stopbuffer
```

Now that all the content elements for the booklet have been defined, the positioning code that follows, once written, need not be changed.

```
% -- Basically you do not need to edit the lines below
```

The CD-booklet is typeset on a custom size of paper. We define this paper size and put it on landscape A4 sheets. We choose our layout parameters and, because we are using the standard-makeup-environment, we turn vertical centering off.

```
\definepapersize[CDbooklet][width=120mm,height=118mm]
\setuppapersize[CDbooklet][A4,landscape]
\setuppagenumbering[location={bottom,right},alternative=doublesided]
\setupcolors[state=start]
\setupnarrower[left=.5em]
\setuplayout
  [topspace=2mm,
  backspace=9mm,
  header=0pt,
```

```

    footer=5mm,
    margin=8.5mm,
    margindistance=.5mm,
    width=100mm,
    height=115mm,
    marking=on,
    location=middle]
\setupmakeup[standard][top=,bottom=]

```

The background image on the cover should be a bleeding image, i.e. it should be larger than the crop marks indicate. However, as soon as we use imposition, the image is cropped to the paper size and the bleed is gone. We define the bleed as follows:

```
\definemeasure[bleed][\dimexpr1mm\relax]
```

Now we define a layer which is filled with the background image.

```

\definelaye
  [Background]
  [position=no]

\setlayer
  [Background]
  {\externalfigure
   [\Pageimage]
   [height=\dimexpr118mm+2\measure{bleed},
    width=\dimexpr120mm+2\measure{bleed}]}

```

The filled layer is placed into the page background. Because we bled the image, we have to add a background offset equal to the bleed.

```
\setupbackgrounds[page][background=Background,
                        backgroundoffset=\measure{bleed}]
```

As mentioned earlier, the booklet is typeset with imposition.

```
\setuparranging[2UP]
```

Now that everything is in place, we can produce the booklet.

```

\starttext

\startstandardmakeup[doublesided=no,page=yes]
  \inleft{\rotate[rotation=90]{%
    \framed[frame=off,align={lohi,middle},width=\textheight]
      {\bfd \getbuffer[Sidetitle]}}}
  \getbuffer[Covertext]
\stopstandardmakeup

\setupbackgrounds[page][background=]
\getbuffer[Bookletcontent]
\stoptext

```

These parameters will produce the following (only the cover page is shown):

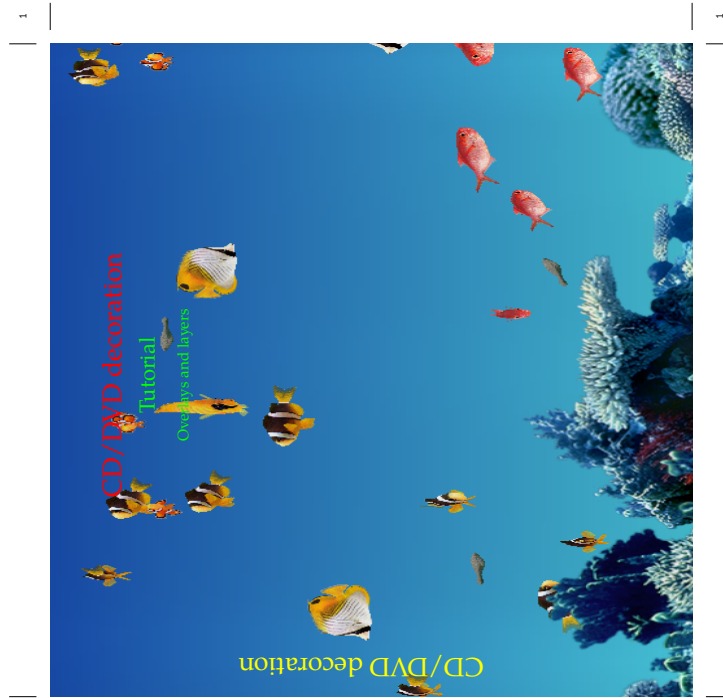


Figure 2. The cover of the booklet

Jewel case inlay

To complete the CD project, we want to prepare an inlay for the jewel case.

As in the previous sections, we start with setting the main language:

```
\mainlanguage[en]
```

We tell ConTeXt the font we want to use for the inlay

```
\usetypscript[palatino]
\setupbodyfont[palatino,10pt]
```

The inlay will be defined such that you can have as many as three images on the inlay. There is a page image, a text image and an image for the small strips to the left and right of the inlay. Again, we define macros in order to replace the actual filename in the code with a symbolic name.

```
\def\Pageimage{fish}
\def\Textimage{eurotexlogo}
\def\SideTitleimage{}
```

We have three text areas. At the left and right sides of the inlay there are small strips along the spine for the title information. Both strips are automatically filled with the same information. In between these is the main text area.

```
\startbuffer[Sidetitle]
  \tfa \yellow CD/DVD decoration\hfill2009
\stopbuffer
```

```
\startbuffer[Maintext]
  \startalignment[middle]
  \startlines
    {\tfc \red CD/DVD decoration}
    \blank \yellow{\tfb Tutorial}
    \blank Overlays and layers
  \stoptlines
  \stopalignment
\stopbuffer
```

The following comment reminds us that the code below this point is usually left untouched.

```
% -- Basically you do not need to edit the lines below.
```

We must define the inlay paper size ourselves. Its dimensions are 150 × 118 mm and typeset on an A4.

```
\definepapersize[CDinlaycase][width=150mm,height=118mm]
\setuppapersize[CDinlaycase][A4]
```

We define a bleed measure to insure that the page image will cover the entire page.

```
\definemeasure[bleed][\dimexpr1mm\relax]
```

We specify the various layout settings. The width of the main text area is 14 mm smaller than the paper width. We use a backspace of 7 mm, which is filled with the margin + margin distance.

```
\setuplayout
  [topspace=0mm,
  backspace=7mm,
  margin=6.5mm,
  header=0pt,
```

```

    footer=0pt,
    margindistance=.5mm,
    width=136mm,
    height=118mm,
    location=middle,
    marking=on]

\setupcolors[state=start]
\setupmakeup[standard][bottom=, top=]

```

To fit the inlay into the jewel case we have to make two folds. It is important to make these folds accurately. To help, we add a layer with fold marks that extend out into the cut space.

```

\definelaye
  [Foldmarks]
  [position=no,
  height=\dimexpr(\paperheight+10mm),
  width=\dimexpr(\paperwidth+10mm),
  x=0mm,
  y=-8mm]

```

Next we define two layers, one each for the page and text images. We do not want the layers to be positioned relative to the text so we set each position to no.

```

\definelaye
  [Pagebackground]
  [position=no]

\definelaye
  [Textbackground]
  [position=no]

```

As mentioned earlier, we can use an overlay to add an image or a background along the spine. The image could be a previously defined external image, and the background could be a transparent color generated with MetaPost.

```

\defineoverlay
  [SideTitlebackground]
  [{\externalfigure[SideTitleimage][width=\overlaywidth,
  height=\overlayheight]]}

```

or

```

\defineoverlay
  [SideTitlebackground]
  [\useMPgraphic{TransparentBackground}]

```

We define two additional layers intended for the side titles. Again, there is no need for relative positioning.

```

\definelaye
  [SideTitleL]
  [position=no]

\definelaye
  [SideTitleR]
  [position=no]

```

We use MetaPost for the fold marks.

```

\startuniqueMPgraphic{Marks}
  path p;
  pair pt[];
  p := unitsquare xscaled 136mm yscaled 135mm;
  pt[1] := llcorner p;
  pt[2] := point 3.95 of p;
  pt[3] := point 3.05 of p;
  pt[4] := ulcorner p;
  pt[5] := lrcorner p;
  pt[6] := point 1.05 of p;
  pt[7] := point 1.95 of p;
  pt[8] := urcorner p;

  for i= 1 step 2 until 7 :
    draw pt[i]--pt[i+1];
  endfor;
\stopuniqueMPgraphic

```

It is easy to prepare a transparent colored background for an overlay with MetaPost.

```

\startreusableMPgraphic{TransparentBackground}
  path p;
  p:= unitsquare xscaled \overlaywidth yscaled \overlayheight;
  fill p withcolor transparent(1,0.3,yellow);
\stopreusableMPgraphic

```

Now we are ready to fill the layers with their respective content, and assign the layers as page or text backgrounds.

```

\setlayer
  [Foldmarks]{\useMPgraphic{Marks}}
\setlayer
  [Pagebackground]
  {\externalfigure
   [\Pageimage]
   [height=\dimexpr118mm+2\measure{bleed},
    width=\dimexpr150mm+2\measure{bleed}]}
\setlayer
  [Textbackground]
  {\externalfigure[\Textimage][height=\textheight,width=\textwidth]}
\setlayer
  [SideTitleL]
  [x=.5mm,y=.5mm]
  {\rotate[rotation=90]{%
   \framed
   [frame=off,align={right,lohi},
    width=\dimexpr\textheight-1mm,
    background=SideTitlebackground]
   {\bf \getbuffer[Sidetitle]}}}
\setlayer
  [SideTitleR]
  [x=-.5mm,y=.5mm]
  {\rotate[rotation=90]{%
   \framed
   [frame=off,align={right,lohi},
    width=\dimexpr\textheight-1mm,

```

```

background=SideTitlebackground]
{\bf \getbuffer[Sidetitle]}}
\setupbackgrounds
[page]
[background=Pagebackground,backgroundoffset=\measure{bleed}]
\setupbackgrounds[text][background={Foldmarks,Textbackground}]
\setupbackgrounds[text][leftmargin][background=SideTitleL]
\setupbackgrounds[text][rightmargin][background=SideTitleR]

```

What remains to be done is to start a document and add the information for the main text area.

```

\starttext
\strut
\framedtext
[frame=off,
rulethickness=3pt,
offset=10pt,
width=\textwidth,
height=\textheight,
align=middle]
{\getbuffer[Maintext]}
\stoptext

```

The result is as follows:



Figure 3. The inlay for the jewel case

Further reading

- ConT_EXt the manual. Hans Hagen. November 2001, available from <http://www.pragma-ade.com/>
- Metafun. Hans Hagen. January 2002, available from <http://www.pragma-ade.com/>
- It's in the details. Hans Hagen. Spring 2002, available from <http://www.pragma-ade.com/>
- <http://wiki.contextgarden.net/Layers>.

Conclusion

This small project demonstrates ConT_EXt's capability to place information and graphics at specific locations using layers. Once you become familiar with how they work, you will find many more situations where layers can be used in modern design. It is worthwhile having a look at the literature references given above since there is more to be said about layers than can be presented in this project.

Acknowledgements

I would like to thank Michael Guravage for proofreading this article and for all the improvements he has made to it.

Willi Egger
w.egger (at) boede.nl

The language mix

Abstract

During the third ConT_EXt conference that ran in parallel to EuroT_EX 2009 in The Hague we had several sessions where mkiv was discussed and a few upcoming features were demonstrated. The next sections summarize some of that. It's hard to predict the future, especially because new possibilities show up once LuaT_EX is opened up more, so remarks about the future are not definitive.

T_EX

From now on, if I refer to T_EX in the perspective of LuaT_EX I mean “Good Old T_EX”, the language as well as the functionality. Although LuaT_EX provides a couple of extensions it remains pretty close to compatible to its ancestor, certainly from the perspective of the end user.

As most ConT_EXt users code their documents in the T_EX language, this will remain the focus of mkiv. After all, there is no real reason to abandon it. However, although ConT_EXt already stimulates users to use structure where possible and not to use low level T_EX commands in the document source, we will add a few more structural variants. For instance, we already introduced `\startchapter` and `\startitem` in addition to `\chapter` and `\item`.

We even go further, by using key/value pairs for defining section titles, bookmarks, running headers, references, bookmarks and list entries at the start of a chapter. And, as we carry around much more information in the (for T_EX so typical) auxiliary data files, we provide extensive control over rendering the numbers of these elements when they are recalled (like in tables of contents). So, if you really want to use different texts for all references to a chapter header, it can be done:

```
\startchapter
  [label=emcsquare,
   title={About  $e=mc^2$ },
   bookmark={einstein},
   list={About  $e=mc^2$  (Einstein)},
   reference={ $e=mc^2$ }]
... content ...
\stopchapter
```

Under the hood, the mkiv code base is becoming quite a mix and once we have a more clear picture of where we're heading, it might become even more of a hybrid.

Already for some time most of the font handling is done by Lua, and a bit more logic and management might move to Lua as well. However, as we want to be downward compatible we cannot go as far as we want (yet). This might change as soon as more of the primitives have associated Lua functions. Even then it will be a trade off: calling Lua takes some time and it might not pay off at all.

Some of the more tricky components, like vertical spacing, grid snapping, balancing columns, etc. are already in the process of being Luaified and their hybrid form might turn into complete Lua driven solutions eventually. Again, the compatibility issue forces us to follow a stepwise approach, but at the cost of (quite some) extra development time. But whatever happens, the T_EX input language as well as machinery will be there.

MetaPost

I never regret integrating MetaPost support in ConT_EXt and a dream came true when MPLIB became part of LuaT_EX. Apart from a few minor changes in the way text integrates into MetaPost graphics the user interface in mkiv is the same as in mkii. Insofar as Lua is involved, this is hidden from the user. We use Lua for managing runs and conversion of the result to PDF. Currently generating MetaPost code by Lua is limited to assisting in the typesetting of chemical structure formulas which is now part of the core.

When defining graphics we use the MetaPost language and not some T_EX-like variant of it. Information can be passed to MetaPost using special macros (like `\MPcolor`), but most relevant status information is passed automatically anyway.

You should not be surprised if at some point we can request information from T_EX directly, because after all this information is accessible. Think of something `w := texdimen(0)` ; being expanded at the MetaPost end instead of `w := \the\dimen0` ; being passed to MetaPost from the T_EX end.

Lua

What will the user see of Lua? First of all he or she can use this scripting language to generate content. But when making a format or by looking at the statistics printed at the end of a run, it will be clear that Lua is used all over the place.

So how about Lua as a replacement for the \TeX input language? Actually, it is already possible to make such “Con \TeX t Lua Documents” using \mkiv ’s built in functions. Each Con \TeX t command is also available as a Lua function.

```
\startluacode
context.bTABLE {
  framecolor = "blue",
  align= "middle",
  style = "type",
  offset=".5ex",
}
for i=1,10 do
  context.bTR()
  for i=1,20 do
    local r= math.random(99)
    if r < 50 then
      context.bTD {
        background = "color",
        backgroundcolor = "blue"
      }
      context(context.white("%#2i",r))
    else
      context.bTD()
      context("%#2i",r)
    end
  end
  context.eTD()
end
context.eTR()
end
context.eTABLE()
\stopluacode
```

Of course it helps if you know Con \TeX t a bit. For instance we can as well say:

```
if r < 50 then
  context.bTD {
    background = "color",
    backgroundcolor = "blue",
    foregroundcolor = "white",
  }
else
  context.bTD()
end
context("%#2i",r)
context.eTD()
```

And, knowing Lua helps as well, since the following is more efficient:

```
\startluacode
local colored = {
  background = "color",
```

```
  backgroundcolor = "bluegreen",
  foregroundcolor = "white",
}
local basespec = {
  framecolor = "blue",
  align= "middle",
  style = "type",
  offset=".5ex",
}
local bTR, eTR = context.bTR, context.eTR
local bTD, eTD = context.bTD, context.eTD
context.bTABLE(basespec)
for i=1,10 do
  bTR()
  for i=1,20 do
    local r= math.random(99)
    bTD((r < 50 and colored) or nil)
    context("%#2i",r)
  end
  eTR()
end
context.eTABLE()
\stopluacode
```

Since in practice the speedup is negligible and the memory footprint is about the same, such optimizations seldom make sense.

At some point this interface will be extended, for instance when we can use \TeX ’s main (scanning, parsing and processing) loop as a so-called coroutine and when we have opened up more of \TeX ’s internals. Of course, instead of putting this in your \TeX source, you can as well keep the code at the Lua end.

84	40	78	80	91	20	34	77	28	55	48	63	37	51	95	91	63	72	15	61
2	25	14	80	16	40	13	11	99	22	51	84	61	30	64	52	49	97	29	77
53	77	40	89	29	35	80	91	7	94	53	9	20	66	89	35	7	2	46	7
24	97	90	85	27	54	38	76	51	67	53	4	44	93	93	72	29	74	64	36
69	17	44	88	83	33	23	89	35	68	95	59	66	86	44	92	40	81	68	91
48	22	95	92	15	88	64	43	62	28	78	31	45	23	19	28	56	42	17	90
11	13	50	76	98	93	68	38	75	37	30	23	58	25	16	73	13	79	17	74
8	95	6	52	18	24	79	73	65	96	64	76	10	14	52	8	7	21	46	82
57	75	6	16	99	21	89	13	99	6	87	8	1	92	59	18	17	39	91	82
36	55	58	45	69	10	53	75	31	99	58	87	75	63	4	75	83	92	87	83

Figure 1. The result of the displayed Lua code.

The script that manages a Con \TeX t run (also called context) will process files with that consist of such commands directly if they have a `cld` suffix or when you provide the flag `--forcecld`.¹

```
context yourfile.cld
```

But will this replace \TeX as an input language? This is quite unlikely because coding documents in \TeX is so convenient and there is not much to gain here. Of course in a pure Lua based workflow (for instance publishing information from databases) it would be nice to code in Lua, but even then it's mostly syntactic sugar, as \TeX has to do the job anyway. However, eventually we will have a quite mature Lua counterpart.

XML

This is not so much a programming language but more a method of tagging your document content (or data). As structure is rather dominant in XML, it is quite handy for situations where we need different output formats and multiple tools need to process the same data. It's also a standard, although this does not mean that all documents you see are properly structured. This in turn means that we need some manipulative power in Con \TeX t, and that happens to be easier to do in \mkiv than in \mkii .

In Con \TeX t we have been supporting XML for a long time, and in \mkiv we made the switch from stream based to tree based processing. The current implementation is mostly driven by what has been possible so far but as Lua \TeX becomes more mature, bits and pieces will be reimplemented (or at least cleaned up and brought up to date with developments in Lua \TeX).

One could argue that it makes more sense to use XSLT for converting XML into something \TeX , but in most of the cases that I have to deal with much effort goes into mapping structure onto a given layout specification. Adding a bit of XML to \TeX mapping to that directly is quite convenient. The total amount of code is probably smaller and it saves a processing step.

We're mostly dealing with education-related documents and these tend to have a more complex structure than the final typeset result shows. Also, readability of code is not served with such a split as most mappings look messy anyway (or evolve that way) due to the way the content is organized or elements get abused.

There is a dedicated manual for dealing with XML in \mkiv , so we only show a simple example here. The documents to be processed are loaded in memory and serialized using setups that are associated to elements. We keep track of documents and nodes in a way that permits multipass data handling (rather usual in \TeX). Say that we have a document that contains questions. The following definitions will flush the (root element) questions:

```
\startxmlsetups xml:mysetups
  \xmlsetsetup{#1}{questions}{xml:questions}
\stopxmlsetups

\xmlregistersetup{xml:mysetups}
```

```
\startxmlsetups xml:questions
  \xmlflush{#1}
\stopxmlsetups

\xmlprocessfile{main}{somefile.xml}{}
```

Here the #1 represents the current XML element. Of course we need more associations in order to get something meaningful. If we just serialize then we have mappings like:

```
\xmlsetsetup{#1}{question|answer}{xml:*}
```

So, questions and answers are mapped onto their own setup which flushes them, probably with some numbering done at the spot.

In this mechanism Lua is sort of invisible but quite busy as it is responsible for loading, filtering, accessing and serializing the tree. In this case \TeX and Lua hand over control in rapid succession.

You can hook in your own functions, like:

```
\xmlfilter{#1}
  {(wording|feedback|choice)/function(cleanup)}
```

In this case the function `cleanup` is applied to elements with names that match one of the three given.²

Of course, once you start mixing in Lua in this way, you need to know how we deal with XML at the Lua end. The following function show how we calculate scores:

```
\startluacode
function xml.functions.totalscore(root)
  local n = 0
  for e in xml.collected(root,"/outcome") do
    if xml.filter(e,"action[text()='add']") then
      local m = xml.filter
        (e,"xml:///score/text()")
      n = n + (tonumber(m or 0) or 0)
    end
  end
  tex.write(n)
end
\stoptluacode
```

You can either use such a function in a filter or just use it as a \TeX macro:

```
\startxmlsetups xml:question
  \blank
  \xmlfirst{#1}{wording}
  \startitemize
    \xmlfilter{#1}
    {/answer/choice/command(xml:answer:choice)}
  \stopitemize
```



Figure 2. An example of using the font tester.

```

\endgraf
score: \xmlfunction{#1}{totalscore}
\blank
\stopxmlsetups

```

```

\startxmlsetups xml:answer:choice
  \startitem
    \xmlflush{#1}
  \stopitem
\stopxmlsetups

```

The filter variant is like this:

```
\xmlfilter{#1}{./function('totalscore')}
```

So you can take your choice and make your source look more XML-ish, Lua-like or \TeX -wise. A careful reader might have noticed the peculiar `xml://` in the function code. When used inside `MkIV`, the serializer defaults to \TeX so results are piped back into \TeX . This prefix forced the regular serializer which keeps the result at the Lua end.

Currently some of the XML related modules, like `MATHML` and handling of tables, are really a mix of \TeX code and Lua calls, but it makes sense to move them completely to Lua. One reason is that their input (formulas and table content) is restricted to non- \TeX

anyway. On the other hand, in order to be able to share the implementation with \TeX input, it also makes sense to stick to some hybrid approach. In any case, more of the calculations and logic will move to Lua, while \TeX will deal with the content.

A somewhat strange animal here is `xsl-fo`. We do support it, but the `MkII` implementation was always somewhat limited and the code was quite complex. So, this needs a proper rewrite in `MkIV`, which will happen indeed. It's mostly a nice exercise of hybrid technology but until now I never really needed it. Other bits and pieces of the current XML goodies might also get an upgrade.

There is already a bunch of functions and macros to filter and manipulate XML content and currently the code involved is being cleaned up. What direction we go also depends on users' demands. So, with respect to XML you can expect more support, a better integration and an upgrade of some supported XML related standards.

Tools

Some of the tools that ship with `ConTeXt` are also examples of hybrid usage.

Take this:

```
mtxrun --script server --auto
```

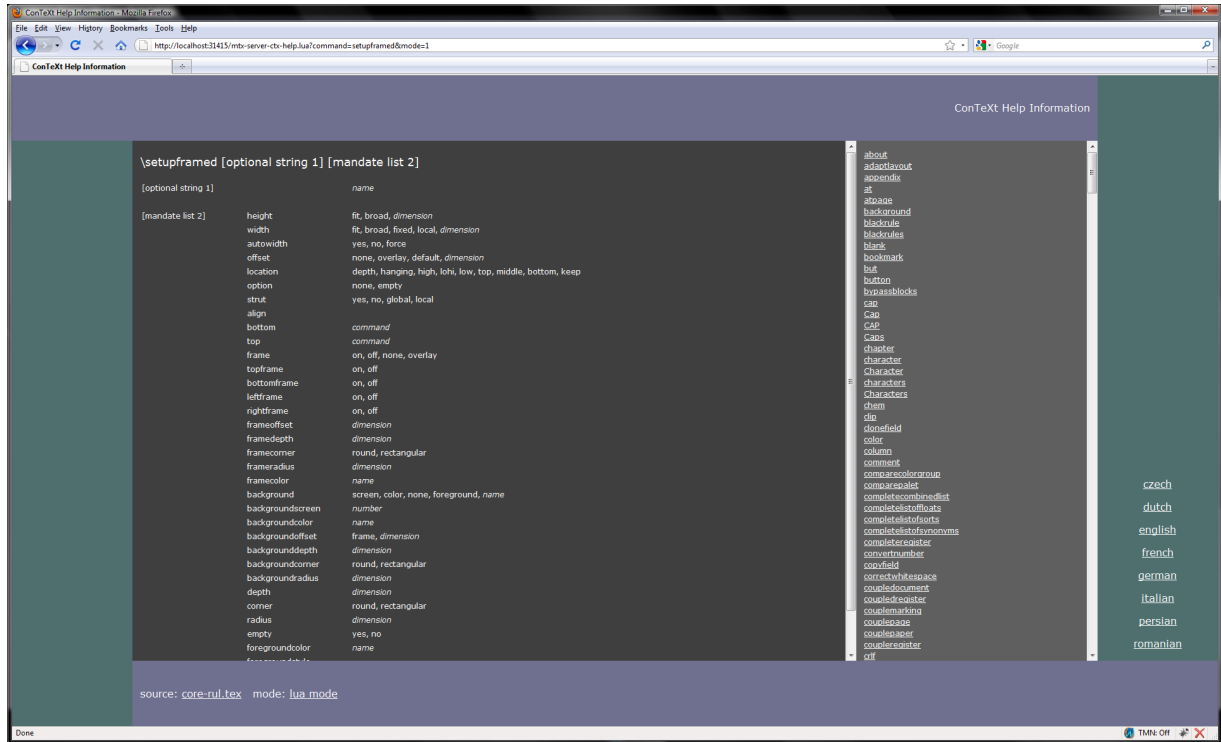


Figure 3. An example of a help screen for a command.

On my machine this reports:

```
MTXrun | running at port: 31415
MTXrun | document root: c:/data/develop/context/
                               lua
MTXrun | main index file: unknown
MTXrun | scripts subpath: c:/data/develop/context
                               /lua
MTXrun | context services: http://localhost:31415
                               /mtx-server-ctx-startup.lua
```

The `mtxrun` script is a Lua script that acts as a controller for other scripts, in this case `mtx-server.lua` that is part of the regular distribution. As we use Lua \TeX as a Lua interpreter and since Lua \TeX has a socket library built in, it can act as a web server, limited but quite right for our purpose.³

The web page that pops up when you enter the given address lets you currently choose between the Con \TeX Xt help system and a font testing tool. In figure 2 you seen an example of what the font testing tool does.

Here we have Lua \TeX running a simple web server but it's not aware of having \TeX on board. When you click on one of the buttons at the bottom of the screen, the server will load and execute a script related to the request and in this case that script will create a \TeX file and call Lua \TeX with Con \TeX Xt to process that file. The

result is piped back to the browser.

You can use this tool to investigate fonts (their bad and good habits) as well as to test the currently available OpenType functionality in MKIV (bugs as well as goodies).

So again we have a hybrid usage although in this case the user is not confronted with Lua and/or \TeX at all. The same is true for the other goodie, shown in figure 3. Actually, such a goodie has always been part of the Con \TeX Xt distribution but it has been rewritten in Lua.

The Con \TeX Xt user interface is defined in an XML file, and this file is used for several purposes: initializing the user interfaces at format generation time, typesetting the formal command references (for all relevant interface languages), for the wiki, and for the mentioned help goodie.

Using the mix of languages permits us to provide convenient processing of documents that otherwise would demand more from the user than it does now. For instance, imagine that we want to process a series of documents in the so-called EPUB format. Such a document is a zipped file that has a description and resources. As the content of this archive is prescribed it's quite easy to process it:

```
context --ctx=x-epub.ctx yourfile.epub
```

This is equivalent to:

```
texlua mtxrun.lua --script context --ctx=x-epub.
                    ctx yourfile.epub
```

So, here we have Lua \TeX running a script that itself (locates and) runs a script context. That script loads a Con \TeX t job description file (with suffix ctx). This file tells what styles to load and might have additional directives but none of that has to bother the end user. In the automatically loaded style we take care of reading the XML files from the zipped file and eventually map the embedded HTML like files onto style elements and produce a PDF file. So, we have Lua managing a run and MKIV managing with help of Lua reading from zip files and converting XML into something that \TeX is happy with. As there is no standard with respect to the content itself, i.e. the rendering is driven by whatever kind of structure is used and whatever the CSS file is able to map it onto, in practice we need an additional style for this class of documents. But anyway it's a good example of integration.

The future

Apart from these language related issues, what more is on the agenda? To mention a few integration related thoughts:

- At some point I want to explore the possibility to limit processing to just one run, for instance by doing trial runs without outputting anything but still col-

lecting multipass information. This might save some runtime in demanding workflows especially when we keep extensive font loading and image handling in mind.

- Related to this is the ability to run MKIV as a service but that demands that we can reset the state of Lua \TeX and actually it might not be worth the trouble at all given faster processors and disks. Also, it might not save much runtime on larger jobs.
- More interesting can be to continue experimenting with isolating parts of Con \TeX t in such a way that one can construct a specialized subset of functionality. Of course the main body of code will always be loaded as one needs basic typesetting anyway.

Of course we keep improving existing mechanisms and improve solutions using a mix of \TeX and Lua, using each language (and system) for what it can do best.

Notes

1. Similar methods exist for processing XML files.
2. This example is inspired by one of our projects where the cleanup involves sanitizing (highly invalid) HTML data that is embedded as a CDATA stream, a trick to prevent the XML file to be invalid.
3. This application is not intentional but just a side effect.

Hans Hagen
 Pragma ADE, Hasselt
 pragma (at) wxs dot nl

E16 & DEtool

typesetting language data using ConT_EXt

Abstract

This article describes two recent projects in which ConT_EXt was used to typeset language data. The goal of project E16 was to typeset the 16th edition of the *Ethnologue*, an encyclopaedia of the languages of the world. The complexity of the data and the size of the project made this an interesting test case for the use of T_EX and ConT_EXt. The Dictionary Express tool (DEtool) is developed to typeset linguistic data in a dictionary layout. DEtool (which is part of a suite of linguistic software) uses ConT_EXt for the actual typesetting.

Introduction

Some background: SIL is an NGO dedicated to serve the world's minority language communities in a variety of language-related ways. Collecting all sorts of language data is the basis of much of the work. This could be things like the number of speakers of a particular language, relations between different languages, literacy rates and bi- and multilingualism. Much of this data ends up in a huge database, which in turn is used as the source for publications like the *Ethnologue*.¹ which is an encyclopaedia of languages. It consists of four parts, starting with an introductory chapter explaining the scope of the publication and 25 pages of 'Statistical summaries'. Part 1 has 600 pages with language descriptions, describing all the 6909 languages of the world. Part 2 consists of 200 pages with language maps and Part 3 has of 400 pages of indexes, for Language names, Language Codes and Country names.

Typesetting the *Ethnologue*

Data flow and directory structure: All the data is stored in an Oracle database running on a secure web server. The XML output is manipulated using XSLT to serve different 'views'. One output path leads to html (for the website <http://www.ethnologue.com>) and another output path gives T_EX-output of with the codes are defined in ConT_EXt. Once the data is downloaded from the server, it is stored locally in the 'data' directory of the typesetting system. There is also a 'content' directory containing small files that \input the data files (and do some tricky things with catcodes.) All the content-files are loaded using a 'project' file in the root directory. This (slightly complicated) process allows for easy updating of the data and convenient testing of all the different parts, both separately and together. The macro definitions are all stored in a module.

Module

In good ConT_EXt style all the code for this project is placed in a module. A ConT_EXt module starts with a header like this:

```
%D \module
%D [      file=p-ethnologue,
%D      version=2009.01.14
%D      title=\CONTEXT\ User Module,
%D      subtitle=Typesetting Ethnologue 16,
%D      author=Jelle Huisman, SIL International,
%D      date=\currentdate,
%D      copyright=SIL International]
%C Copyright SIL International
```



```
\writestatus{loading}{Context User Module Typesetting Ethnologue 16}
\unprotect
\startmodule[ethnologue]
```

All the macro definitions go here... and the module is closed with:

```
\stopmodule
\protect \endinput
```

With the command `texexec --modu p-ethnologue.tex` it is easy to make a pdf with the module code, comments and even an index.

E16 code examples

A couple of code examples are presented here to give an impression of the project. This is part of the standard page setup for the paper size and the setup of two basic layouts.

```
\definepapersize [ethnologue][width=179mm, height=255mm]

\startmode[book] % basic page layout for the book
\setuppapersize [ethnologue][letter]% paper size for book mode
\setuplayout[backspace=18mm, width=148mm, topspace=7mm, top=0mm,
             header=6mm, footer=7mm, height=232mm]
\stopmode

\startmode[proofreading] % special layout for proofreading mode
\setuppapersize [letter][letter]% paper size for proofreading mode
\setuplayout[backspace=18mm, width=160mm, topspace=7mm, top=0mm,
             header=16mm, footer=6mm, height=250mm]
\stopmode
```

Use of modes: proofreading vs. final output

To facilitate the proofreading a special proofreading ‘mode’ was defined with wider margins, as shown in the code example in the previous section and with a single column layout (not in this code example). The ‘modes’ mechanism is used to switch between different setups. This code:

```
%\enablemode[book]
\enablemode[proofreading]
```

is used in a ‘project setup’ file to switch between the proofreading mode (single column, bigger type) and the book mode showing the layout of the final publication. One other application of modes is the possible publication of separate extracts with e.g. the language descriptions of only one country. This could be published using a Printing on Demand process.

Language description

The biggest part of the publication is the section with the language descriptions. Each language description consists of: a page reference (not printed), the language name, the language code, a short language description and a couple of special ‘items’ like: language class, dialects, use and writing system. This is an example of the raw data for Belarusian:

```
\startLaDes{ % start of Language Description
\pagereference[bel-BY] % used for index
\startLN{Belarusian }\stopLN % LN: Language name
[bel] % ISO 639-3 code for this language
(Belarusian, Belorussian, Bielorrussian, Byelorussian, White Russian,
White Ruthenian). 6,720,000 in Belarus (Johnstone and Mandryk 2001).
Population total all countries: 8,620,000. Ethnic population:
9,051,080. Also in Azerbaijan, Canada, Estonia, Kazakhstan,
Kyrgyzstan, Latvia, Lithuania, Moldova, Poland, Russian Federation
```

Sine, Dyegueme (Gyegem), Niominka. The Niominka and Serere-Sine dialects mutually inherently intelligible. *Lg Use*: Official language. National language. *Lg Dev*: Literacy rate in L1: Below 1%. Bible: 2008. *Writing*: Arabic script. Latin script. *Other*: 'Sereer' is their name for themselves. Traditional religion, Muslim, Christian. *Map*: 725:28.

Soninke [snk] (Marka, Maraka, Sarahole, Sarakole, Sarangkolle, Sarawule, Serahule, Serahuli, Silabe, Toubakai, Walpre). 250,000 in Senegal (2007 LeClerc). North and south of Bakel along Senegal River. Bakel, Ouaoundé, Moudéri, and Yaféra are principal towns. *Dialects*: Azer (Adjer, Aser), Gadyaga. *Lg Use*: Official language. National language. Also use French, Bambara [bam], or Fula [fub]. *Lg Dev*: Literacy rate in L1: Below 1%. *Other*: The Soninke trace their origins back to the Eastern dialect area of Mali (Kinbakka), whereas the northeastern group in Senegal is part of the Western group of Mali (Xenqenna). Thus, significant differences exist between the dialects of the 2 geographical groups of Soninke in Senegal. Muslim. See main entry under Mali. *Map*: 725:29.

Wamey [cou] (Conhague, Coniagui, Koniagui, Konyagi, Wamei). 18,400 in Senegal (2007), decreasing. Population total all countries: 23,670. Southeast and central along Guinea border, pockets, usually beside Pulaar [fuc]. Also in Guinea. *Class*: Niger-Congo, Atlantic-Congo, Atlantic, Northern, Eastern Senegal-Guinea, Tenda. *Lg Use*: Neutral attitude. Also use Pulaar [fuc]. *Lg Dev*: Literacy rate in L1: Below 1%. *Writing*: Latin script. *Other*: Konyagi is the ethnic name. Agriculturalists; making wine, beer; weaving bamboo mats. Traditional religion, Christian. *Map*: 725:30.

Wolof [wol] (Ouolof, Volof, Walaf, Waro-Waro, Yalof). 3,930,000 in Senegal (2006). Population total all countries: 3,976,500. West and central, Senegal River left bank to Cape Vert. Also in France, Gambia, Guinea-Bissau, Mali, Mauritania. *Class*: Niger-Congo, Atlantic-Congo, Atlantic, Northern, Senegambian, Fula-Wolof, Wolof. *Dialects*: Baol, Cayor, Dyolof (Djolof, Jolof), Lebou (Lebu), Jander. Different from Wolof of Gambia [wof]. *Lg Use*: Official language. National language. Language of wider communication. Main African language of Senegal. Predominantly urban. Also use French or Arabic. *Lg Dev*: Literacy rate in L1: 10%. Literacy rate in L2: 30%. Radio programs. Dictionary. Grammar. NT: 1988. *Writing*: Arabic script, Ajami style. Latin script. *Other*: 'Wolof' is their name for themselves. Muslim. *Map*: 725:32.

Xasonga [kao] (Kasonke, Kasso, Kasson, Kassonke, Khasonke, Xaasonga, Xaasongaxango, Xasonke). 9,010 in Senegal (2006). *Lg Dev*: Literacy rate in L1: Below 1%. *Other*: Muslim. See main entry under Mali (Xaasongaxango).

Seychelles

Republic of Seychelles. 86,000. National or official languages: English, French, Seselwa Creole French. Includes Aldabra, Farquhar, Des Roches; 92 islands. Literacy rate: 62%–80%. Information mainly from D. Bickerton 1988; J. Holm 1989. Blind population: 150 (1982 WCE). The number of individual languages listed for Seychelles is 3. Of those, all are living languages.

English [eng]. 1,600 in Seychelles (1971 census). *Lg Use*: Official language. *Other*: Principal language of the schools. See main entry under United Kingdom.

French [fra]. 980 in Seychelles (1971 census). *Lg Use*: Official language. *Other*: Spoken by French settler families, 'grands blancs'. See main entry under France.

Seselwa Creole French [crs] (Creole, Ilois, Kreol, Seychelles Creole French, Seychellois Creole). 72,700

(1998). Ethnic population: 72,700. *Class*: Creole, French based. *Dialects*: Seychelles dialect reportedly used on Chagos Islands. Structural differences with Morisyen [mfe] are relatively minor. Low intelligibility with Réunion Creole [rcf]. *Lg Use*: Official language since 1977. All domains. Positive attitude. *Lg Dev*: Taught in primary schools. Radio programs. Dictionary. Grammar. NT: 2000. *Writing*: Latin script. *Other*: Fishermen. Christian.

Sierra Leone

Republic of Sierra Leone. 5,586,000. National or official language: English. Literacy rate: 15%. Immigrant languages: Greek (700), Yoruba (3,800). Also includes languages of Lebanon, India, Pakistan, Liberia. Information mainly from D. Dalby 1962; TISL 1995. Blind population: 28,000 (1982 WCE). Deaf institutions: 5. The number of individual languages listed for Sierra Leone is 25. Of those, 24 are living languages and 1 is a second language without mother-tongue speakers. See map on page 726.

Bassa [bsq]. 5,730 in Sierra Leone (2006). Freetown. *Other*: Traditional religion. See main entry under Liberia.

Bom [bmf] (Bome, Bomo, Bum). 5,580 (2006), decreasing. Along Bome River. *Class*: Niger-Congo, Atlantic-Congo, Atlantic, Southern, Mel, Bullom-Kissi, Bullom, Northern. *Dialects*: Lexical similarity: 66%–69% with Sherbro [bun] dialects, 34% with Krim [krm]. *Lg Use*: Shifting to Mende [men]. *Other*: Traditional religion.

Bullom So [buy] (Bolom, Bulem, Bullin, Bullun, Mandenyi, Mandingi, Mmani, Northern Bullom). 8,350 in Sierra Leone (2006). Coast from Guinea border to Sierra Leone River. Also in Guinea. *Class*: Niger-Congo, Atlantic-Congo, Atlantic, Southern, Mel, Bullom-Kissi, Bullom, Northern. *Dialects*: Mmani, Kafu. Bom is closely related. Little intelligibility with Sherbro, none with Krim. *Lg Use*: Shifting to Themne [tem]. *Lg Dev*: Bible portions: 1816. *Writing*: Latin script. *Other*: The people are intermarried with the Temne and the Susu. Traditional religion. *Map*: 726:1.

English [eng]. *Lg Use*: Official language. Used in administration, law, education, commerce. See main entry under United Kingdom.

Gola [gol] (Gula). 8,000 in Sierra Leone (1989 TISLL). Along the border and inland. *Dialects*: De (Deng), Managobla (Gobla), Kongbaa, Kpo, Senje (Sene), Tee (Tege), Toldil (Toodii). *Lg Use*: Shifting to Mende [men]. *Other*: Different from Gola [mzm] of Nigeria (dialect of Mumuye) or Gola [pbp] (Badyara) of Guinea-Bissau and Guinea. Muslim, Christian. See main entry under Liberia. *Map*: 726:4.

Kisi, Southern [kss] (Gissi, Kisi, Kissien). 85,000 in Sierra Leone (1995). *Lg Dev*: Literacy rate in L2: 3%. *Other*: Different from Northern Kisi [kqs]. Traditional religion, Muslim, Christian. See main entry under Liberia. *Map*: 726:13.

Kissi, Northern [kqs] (Gizi, Kisi, Kisie, Kissien). 40,000 in Sierra Leone (1991 LBT). *Dialects*: Liaro, Kama, Teng, Tung. *Lg Use*: Also use Krio [kri] or Mende [men]. *Other*: Traditional religion. See main entry under Guinea. *Map*: 726:11.

Klao [klu] (Klaoh, Klau, Kroo, Kru). 9,620 in Sierra Leone (2006). Freetown. Originally from Liberia. *Other*: Traditional religion. See main entry under Liberia.

Kono [kno] (Konnoh). 205,000 (2006). Northeast. *Class*: Niger-Congo, Mandé, Western, Central-Southwestern, Central, Manding-Jogo, Manding-Vai, Vai-Kono. *Dialects*: Northern Kono (Sando), Central Kono (Fiama, Gbane, Gbane Kando, Gbense, Gorama Kono, Kamara, Lei, Mafindo, Nimi Koro, Nimi Yama, Penguia, Soa, Tankoro,

Figure 1. Example of page with language descriptions

```
(Europe), Tajikistan, Turkmenistan, Ukraine, United States, Uzbekistan.
\startLDitem{Class: }\stopLDitem % LDitem: Language description item
Indo-European, Slavic, East.
\startLDitem{Dialects: }\stopLDitem Northeast Belarusan (Polots,
Viteb-Mogilev), Southwest Belarusan (Grodnen-Baranovich,
Slutsko-Mozyr, Slutska-Mazyrski), Central Belarusan. Linguistically
between Russian and Ukrainian [ukr], with transitional dialects to both.
\startLDitem{Lg Use: }\stopLDitem National language.
\startLDitem{Lg Dev: }\stopLDitem Fully developed. Bible: 1973.
\startLDitem{Writing: }\stopLDitem Cyrillic script.
\startLDitem{Other: }\stopLDitem Christian, Muslim (Tatar). }
\stopLaDes % end of Language Description
```

The styles for the different elements are defined using start-stop setups. One example is the style for the LDitem (Language Definition item) which was initially coded in this way:

```
\definestartstop % Language Description Item Part 1 % deprecated code!
[LDitem]
[before={\switchtobodyfont[GentiumBookIt,\LDitemfontsize]},
after={\switchtobodyfont[Gentium,\bodyfontpartone]}]
```

Eventually bodyfont switches were replaced by proper ConT_EXt-style typescripts, but the idea remains the same: `\definestartstop[something][code here]` makes it possible to use the pair `\startsomething` and `\stopsomething`.

Dynamic running header

As the example of the page with language descriptions (figure 1) shows the Country name is inserted in the header of the page, using the first country on a left page and the last country on the right page. The code used to do this is based on an example in `page-set.tex` in the ConT_EXt distribution.

```
\definemarking[headercountryname]
\setupheadertexts[\setups{show-headercountryname-marks}]
\startsetups show-headercountryname-first
\getmarking[headercountryname][1][first] % get first marking
\stopsetups
\startsetups show-headercountryname-last
\getmarking[headercountryname][2][last] % get last marking
\stopsetups
\setupheadertexts[]
\setupheadertexts
[\setups{text a}][]
[[][\setups{text b}]] % setup header text (left and right pages)
\startsetups[text a] % setup contents page a
\rlap{Ethnologue}
\hfill
{\pagenumber}
\hfill
\llap{\setups{show-headercountryname-last}}
\stopsetups
\startsetups[text b] % setup contents page b
\rlap{\setups{show-headercountryname-first}}
\hfill
\pagenumber
\hfill
\llap{Ethnologue}
\stopsetups
```

Language Name Index

This index lists every name that appears in Part I as a primary or alternate name of a language or dialect. The following abbreviations are used in the index entries: *alt.* ‘alternate name for’, *alt. dial.* ‘alternate dialect name for’, *dial.* ‘primary dialect name for’, *pej. alt.* ‘pejorative alternate name for’, and *pej. alt. dial.* ‘pejorative alternate dialect name for’. The index entry gives the primary name for the language

with which the given name is associated, followed by the unique three-letter language code in square brackets. The numbers identify the pages on which the language entries using the indexed name may be found. If the list of page references includes the entry in the primary country, it is listed first. The entry for a primary name also lists page numbers for the maps on which the language occurs.

- A Fala de Xálima**, *alt.* Fala [fax], 575
A Fala do Xálima, *alt.* Fala [fax], 575
A Nden, *alt.* Abun [kgr], 427
'A Vo', *alt. dial.* Awa [vwa], 335
'A vo' loi, *alt. dial.* Awa [vwa], 335
A'a Sama, *alt. dial.* Sama, Southern [ssb], 473
Aachterhoeks, *alt.* Achterhoeks [act], 563
Aage, *alt.* Esimbi [ags], 70, 171
Aaimasa, *alt. dial.* Kunama [kun], 121
Aal Murrah, *alt. dial.* Arabic, Najdi Spoken [ars], 523
Aalan, *alt.* Allar [all], 366
Aalawa, *dial.* Ramoovina [rai], 633
Aalawaa, *alt. dial.* Ramoovina [rai], 633
Aaleira, *alt.* Laro [lro], 204
Aantantara, *dial.* Tairora, North [tbg], 637
A'ara, *alt.* Cheke Holo [mrn], 646
Aarai, *alt.* Aari [aiw], 121
Aari [aiw], 121, 699
Aariya [aay], 365
Aasá, *alt.* Aasáx [aas], 207
Aasáx [aas], 207, 731
Aatasaara, *dial.* Tairora, South [omw], 637
AAVE, *alt. dial.* English [eng], 310
AAye, *alt. dial.* Shua [shg], 58
Aba, *alt.* Amba [utp], 645
alt. Shor [cjs], 522
dial. Tibetan [bod], 404
Abá, *alt.* Avá-Canoeiro [avv], 237
Abangi, *alt. dial.* Gwamhi-Wuri [bga], 173
Ababda, *dial.* Bedawiyet [bej], 121
Abaca, *alt. dial.* Ilongot [ilk], 511
Abacama, *alt.* Bacama [bcy], 165
Abacha, *alt.* Basa [bzw], 166
Abadani, *dial.* Farsi, Western [pes], 454
Abadhi, *alt.* Awadhi [awa], 484
Abadi [kbt], 600, 877
alt. Awadhi [awa], 367, 484
alt. Tsuvadi [tvd], 187
Abadzakh, *alt. dial.* Adyghe [ady], 567
Abadzeg, *alt. dial.* Adyghe [ady], 567
Abadzex, *dial.* Adyghe [ady], 567
Abaga [abg], 600, 871
Abai, *dial.* Putoh [put], 411
Abai Sungai [abf], 471, 811
Abak, *dial.* Anaang [anw], 165
Abaka, *dial.* Ilongot [ilk], 511
Abakan, *alt.* Kpan [kpk], 178
Abakan Tatar, *alt.* Khakas [kjh], 520, 345
Abakay Spanish, *alt. dial.* Chavacano [cbk], 509
Abaknon, *alt.* Inabaknon [abx], 511
Abaknon Sama, *alt.* Inabaknon [abx], 511
Abakoum, *alt.* Kwakum [kwu], 74
Abakpa, *alt. dial.* Ejagham [etu], 170, 70
Abakum, *alt.* Kwakum [kwu], 74
Abakwariga, *alt.* Hausa [hau], 173
Abaletti, *dial.* Yele [yle], 644
Abam, *dial.* Wipi [gdr], 642
Abancay, *dial.* Quechua, Eastern Apurímac [qve], 300
Abane, *alt.* Baniva [bvv], 320
Abangba, *alt.* Bangba [bbe], 106
Abanliku, *alt.* Obanliku [bzy], 183
Abanyai, *alt. dial.* Kalanga [kck], 227
Abanyom [abm], 164, 724
Abanyum, *alt.* Abanyom [abm], 164
Abar [mij], 65, 685
Abarambo, *alt.* Barambu [brm], 106
Abasakur, *alt.* Pal [abw], 632
Abathwa, *alt.* Xegwi [xeg], 198
Abatonga, *alt. dial.* Ndaou [ndc], 228
Abatsa, *alt.* Basa [bzw], 166
Abau [aau], 601, 866
Abaw, *alt.* Bankon [abb], 67
Abawa, *dial.* Gupa-Abawa [gpa], 173
Abayongo, *dial.* Agwagwune [yay], 164
Abaza [abq], 567, 533, 849
Abazin, *alt.* Abaza [abq], 567, 533
Abazintsy, *alt.* Abaza [abq], 567, 533
Abbé, *alt.* Abé [aba], 100
Abbey, *alt.* Abé [aba], 100
Abbey-Ve, *dial.* Abé [aba], 100
Abbruzzesi, *dial.* Romani, Sinte [rmo], 572
'Abd Al-Kuri, *dial.* Soqotri [sq], 543
Abdal, *alt.* Ainu [aib], 335
Abdedal, *alt.* Gagadu [gbu], 584
Abe, *dial.* Anyin [any], 100
Abé [aba], 100, 692
Abedju-Azaki, *dial.* Lugbara [lgg], 112
Àbéélé, *alt.* Beele [bxq], 166
Abefang, *alt. dial.* Befang [bby], 68
Abelam, *alt.* Ambulas [abt], 602
Abellen Ayta, *see* Ayta, Abellen [abp], 507
Abenaki, *alt.* Abnaki, Eastern [aaq], 306
alt. Abnaki, Western [abe], 247
Abenaqui, *alt.* Abnaki, Western [abe], 247
Abendago, *alt.* Yali, Pass Valley [yac], 441
Abeng, *dial.* Garo [grt], 329
A'beng, *dial.* Garo [grt], 375
A'bengya, *alt. dial.* Garo [grt], 375
Abenlen, *alt.* Ayta, Abellen [abp], 507
Aberu, *dial.* Mangbetu [mdj], 113
Abewa, *alt.* Asu [aum], 165
Abgue, *dial.* Birgit [btf], 88
Abhor, *alt.* Adi [adi], 365
Abi, *alt.* Abé [aba], 100
Abia, *alt.* Aneme Wake [aby], 602
Abiddul, *alt.* Gagadu [gbu], 584
Abidji [abi], 100, 692
Abie, *alt.* Aneme Wake [aby], 602
Abiem, *dial.* Dinka, Southwestern [dik], 201
Abigar, *alt. dial.* Nuer [nus], 126
dial. Nuer [nus], 205
Abigira, *alt.* Abishira [ash], 295
Abiji, *alt.* Abidji [abi], 100
Abiliang, *dial.* Dinka, Northeastern [dip], 201
Abini, *dial.* Agwagwune [yay], 164
Abinomn [bsa], 427, 797
Abinsi, *alt.* Wannu [jub], 188
Abipon [axb], 231
Abiquira, *alt.* Abishira [ash], 295
Abira, *alt.* E'ñapa Woromaipu [pbh], 320
Abiri, *alt.* Mararit [mgb], 92

Index

Since all the data for this publication comes from a database it was easy to compile a list of index items from that data. Page numbers were resolved using ConT_EXt's internal referencing system. The data contains references using three letter ISO code for language and a two letter country code like this:

```
\pagereference[bel-BY] % ISO code - country code
```

In the file with the index data this reference is linked to an index item:

```
Belarusan [bel], \at[bel-BY]
```

The code [bel-BY] is automatically replaced by the right page number(s) producing the correct entry in the index:

```
Belarusan [bel], 32, 224
```

Since the language name index (the biggest index) contains more than 100.000 references it can be imagined that typesetting this publication in one run was pushing the limits of T_EX. This is the first time that ConT_EXt is used to typesetting this publication. The previous version was produced using Ventura but when that program was replaced by InDesign there were some questions about the way in which InDesign works with the automatically generated data. T_EX seemed to be the right tool to use for this project and it sparked renewed interest in the use of T_EX for other data-intensive publications like dictionaries.

Exploring language

Counting languages is not the only way to collect language data: many linguists move into a language group and take a closer look at the different parts of the actual languages. Some linguists focus on the sounds of a language, others analyse the sentence structure or the way in which language is used in specific communication processes. The collected data is stored in a special database program called FieldWorks. FieldWorks runs on Windows only (though a Linux port is work in progress) and it is a free download from the SIL website². FieldWorks is actually a suite of programs consisting of Data Notebook, Language Explorer and WorldPad. FieldWorks Data Notebook is used for anthropological observations. FieldWorks WorldPad is a 'world ready' text editor with some special script support (including Graphite³). FieldWorks Language Explorer (FLE_x) is used to store all sorts of language related data. It is basically a complex database program with a couple of linguistics related tools. FLE_x contains a lexicon for storing data related to words, meaning(s), grammatical information about words and translations in other languages. Another part of FLE_x is the interlinear tool which makes it possible to take a text in one language and to give a 'word for word translation' in another language, for example as a way to discover grammatical structures. FLE_x comes with a grammar tool to facilitate the analysis and description of the grammar of a language. Since all language data is stored in the same database there are some interesting possibilities to integrate the language data and analysis tools.

Dictionaries

Once a field linguist has collected a certain amount of data he can start to think about the production of a word list or a real dictionary. To facilitate this a team of programmers has made tool called 'Dictionary Express'. This tool allows for the easy production of dictionaries based on data available in the FLE_x database. The user of FLE_x gets a menu option 'Print dictionary' and is presented with small window to enter some layout options. Behind the scenes one of two output paths is used: one is based on the use of an OpenOffice document template and another one uses X_YT_EX and ConT_EXt to typeset the dictionary. X_YT_EX was chosen because of the requirement to facilitate the

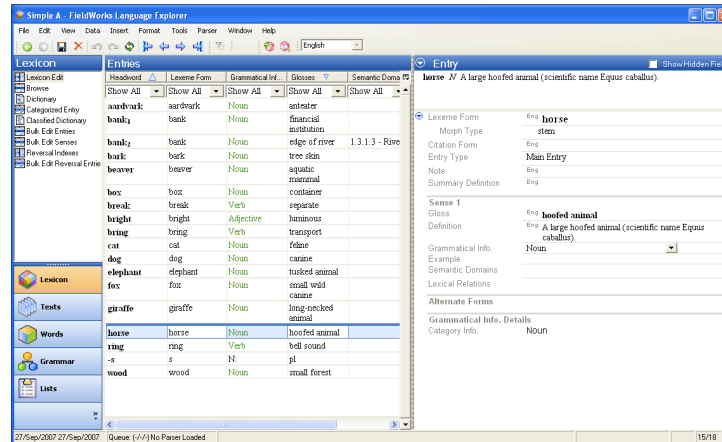


Figure 3. FieldWorks Language Explorer main window

use of the Graphite smart font technology used for the correct rendering of complex non-roman script fonts in use in some parts of the world (see footnote 2). The use of X_qTeX does of course mean that we use ConT_EXt MkII.

All data is available in an XML format and converted (using a purpose built converter) to a simple T_EX-tagged format. A typical dictionary entry looks like this:

```

\Bentry
\Bhw{abel}\Ehw
\marking[guidewords]{abel}
\Bpr{a.imbɛl}\Epr
\Bps{noun(al)}\Eps
\Bl{Eng}\Elt
\Bde{line, row}\Ede
\Bl{Pdg}\Elt
\Bde{lain}\Ede
\Bps{noun(al)}\Eps
\Bl{Eng}\Elt
\Bde{pole, the lowest of the three horizontal poles to which a fence is
tied and which form the main horizontal framework for the fence. This
is the biggest of the three}\Ede
\Eentry

```

The tags used in this data file include:

- headword (hw): this is the word that this particular entry is about,
- pronunciation (pr): the proper pronunciation of the word written using the International Phonetic Alphabet (IPA),
- part of speech (ps): the grammatical function of the word,
- language tag (lt): the language of the definition or example,
- definition (de): meaning of the headword,
- example (ex): example of the word used in a sentence.
- \marking[guidewords]{}: is used to put the correct guideword at the top of each page. (The code used here is inspired by the code used to put country name in the headers in the Ethnologue project.)

Currently most of the required features are implemented. This includes: font selection (including the use of Graphite fonts), basic dictionary layout and picture support. Some of these features are strait-forward and easy to implement. Other features such as picture support required more work e.g. page wide pictures keep floating to the next

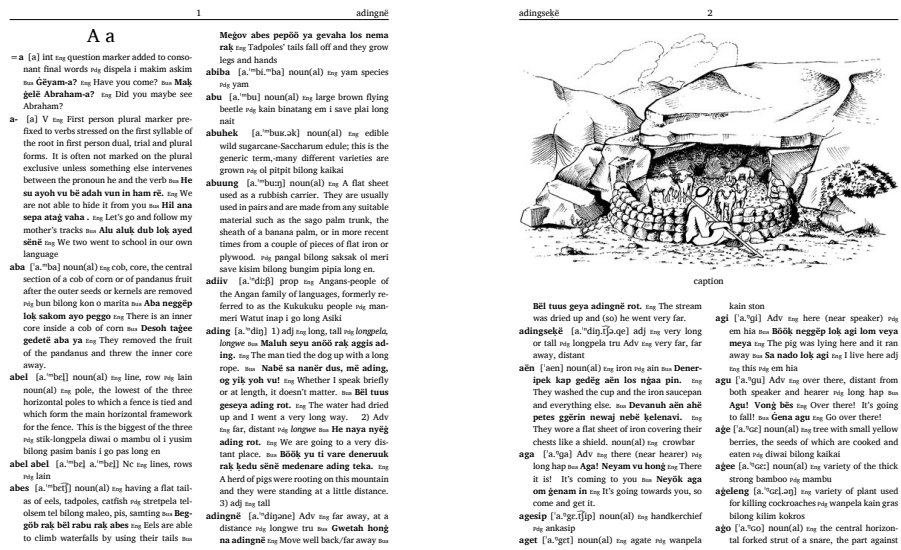


Figure 4. Sample double column dictionary layout

page. Since it is usually a good idea to separate form and content most of the layout related settings are not stored in the data file itself but in a separate settings file which is loaded at the start of the typesetting process. Examples of settings in this file include the fonts and the use of a double column layout. Default settings are used unless the user has specified different settings using the small layout options window at the start of the process.

Currently the test version of this ConT_EXt-based system works with a stand alone ConT_EXt-installation, using the ‘minimals’ distribution. One of the remaining challenges is to make a light weight, easy to install version of ConT_EXt which can be included with the FieldWorks software. Since the main script used by ConT_EXt Mark II is a Ruby script this requires dealing with (removing) the Ruby dependency. It is hoped that stripping the T_EX-tree of all unused fonts and code will help too to reduce the space used by this tool. This is currently work in progress.

Footnotes

1. Lewis, M. Paul (ed.), *Ethnologue: Languages of the World*, Sixteenth edition. Dallas, Tex.: SIL International (2009)
2. <http://www.sil.org/computing/fieldworks/>
3. <http://scripts.sil.org/RenderingGraphite>

Jelle Huisman
 SIL International
 Horsleys Green
 High Wycombe
 United Kingdom
 HP14 3XL
 jelle_huisman (at) sil (dot) org

A network T_EX Live installation at the University of Groningen

Abstract

This article describes a network T_EX Live installation for Windows users and the context in which it operates.

Keywords

T_EX Live, MiK_TE_X, installers, editors, roaming profiles, Windows Vista

Our university has a LAN-based T_EX installation for Windows users. The current edition is based on T_EX Live. After some historical notes, I discuss the computing environment at the university, the new T_EX Live-based installation and issues with Windows Vista.

This article can be considered an update of a previous article¹ about the MiK_TE_X-based installation that I maintained for the university's economics department.

Prehistory: 4T_EX

Our department has had a network T_EX installation for DOS/Windows users since the early nineties. It started out as a simple 4DOS menu for running T_EX and associated programs. Later, it evolved into 4T_EX, and was also made available to members of the Dutch-speaking T_EX users group (the NTG) and others; see figure 1.

The final version was a Windows program, based on the same core as T_EX Live. It included a vast array of utilities, some of them third-party or shareware.

A MiK_TE_X-based installation

When I took over in 2003, 4T_EX was no longer being developed. We chose to make a fresh start, based on what was then available, and to limit ourselves to free software.

Modern L^AT_EX editors such as T_EXnicCenter can take care of running BiB_TE_X and MakeIndex, include spell checkers, and offer help in entering L^AT_EX macros and in debugging. For many users, the editor is all they see from the T_EX installation.

The good integration of MiK_TE_X as the T_EX implementation and T_EXnicCenter as editor and frontend was hard to argue with, so that was what we used.

For graphics support, there was a script to install WMF2EPS and its PostScript printer driver which did the

```

4TEX v3.27

main TeX file      : test .tex
include TeX file   : .tex
TeX files path     : c:\ndocs
backup path        : c:\nbak
TeX format         : big LaTeX Ze + Babel

main menu:
change Main TeX file      choose TeX Format      check Spelling
change Include file       Compile TeX file       run Utilities
change TeX files Path     show Logfile           execute Dos command
change files eXtensions   manage Output           Backup TeX files
Edit file(s)              Quit

press highlighted key or
press [F1] or [Alt] highlighted key for help
press [?] for info on 4TEX

make your choice... _

(c) 4U 1991-1995

```

Figure 1. 4T_EX main menu

real work in the background. For anything else, users had to look elsewhere.

Evolution

Installer. The original installer was a combined batch-file/Perl script, which also used some registry patches. This was replaced with a GUI installer based on NSIS, an open source installation system. Where possible, files and registry settings were generated during installation rather than copied from a prototype installation.

Updated versions. The editor and MiK_TE_X have been updated several times. This included a major change

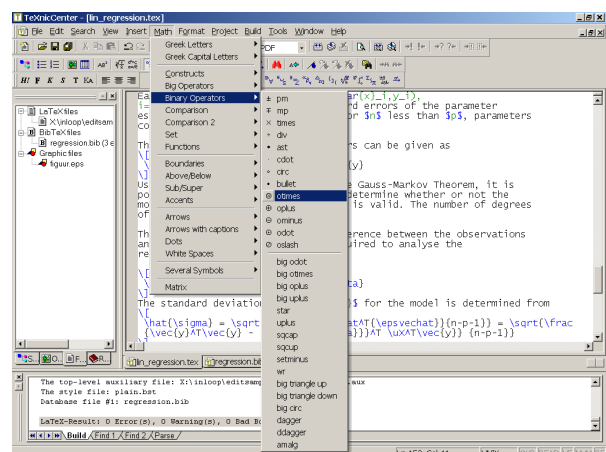


Figure 2. TeXnicCenter as frontend to MiK_TE_X

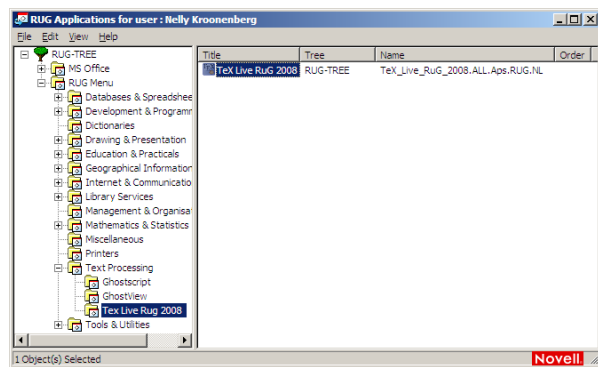


Figure 3. The NAL application menu

in configuration strategy in MiK_TE_X from version 2.4 to version 2.5. I understand that there are again major changes with MiK_TE_X 2.8.

CD edition. A companion CD was created. At first this contained a set of downloaded installers with directions for use. As a result, I was asked time and again to install T_EX on people's laptops. Therefore, a later version got a modified version of the network installer. Nowadays, the CD is offered as an ISO image in the root of the installation.

Expanded user base. The user base for our MiK_TE_X was expanded first with students, then with other departments.

Standardization. Around the time of the second MiK_TE_X edition, we also moved to standardized desktops and roaming profiles; see the next section.

Graphics support. WMF2EPS was dropped, in part because it was shareware and I didn't want to deal with licensing issues for the expanded user base, in part for technical reasons. In its place, I created EPSPDF to take care of converting and cropping arbitrary PostScript (print)files.

The university network

For some time, we have had a centrally managed university-wide Novell network. Software and licenses are also centrally managed. There is a standardized Windows XP workstation. Standard software and additional software is installed from the network and where possible also run from the network. NAL (Novell Application Launcher) is the network component which takes care of this.

Figure 3 displays the NAL menu as seen from my computer at the university (the T_EX Live entry merely points to the installation script).

Staff members can and do install software of their own on their local system if they want to. Students do not

have this luxury. Some staff members download and install their own T_EX.

The standard workstation is configured with *roaming profiles*, i.e. user configuration is mostly copied to the network on logout and copied back to the local system on login. Users see the same desktop on any client computer on the net, of course excepting software which staff members may have installed locally.

Roaming profile configuration should involve nothing local, unless it is copied to and from the network as part of the user profile. It should not require admin rights either. This is especially important for classroom computers and for students.

T_EX Live

In 2008 I got involved in the T_EX Live project. I mostly worked on Windows support, keeping an eye on the needs of our university installation.

I have done only a little work on the 2009 edition. However, other team members now also have Windows virtual machines for testing, and we have been joined by a real Windows user, Tomasz Trzeciak. He proved to us that DOS batchfiles aren't quite as lame as we thought they were.

Compared to MiK_TE_X 2.5, T_EX Live is a lot simpler to turn into a network installation:² in good Unix tradition, T_EX Live uses environment variables and plain text files for configuration.

Relocatable. An important function of configuration is telling programs where they can find the files they need. Normally, T_EX Live puts only relative paths in the configuration files. Programs can combine these relative paths with their own location to determine absolute paths. With this strategy, configuration files can stay the same if the installation as a whole is transferred to another place.

Batteries included. T_EX Live contains copies of Perl and Ghostscript for Windows. This puts Windows on a more equal footing with Unix/Linux with regard to all the scripted utilities that are part of a typical T_EX installation.

Both the included Ghostscript and the included Perl are hidden, i.e. T_EX Live knows that they are there, but the rest of the system doesn't. They are not on the search path, and there are no environment variables or registry settings created for them. Therefore, they shouldn't interfere with pre-installed copies. The only disadvantage is the disk space they take up. But this is hardly significant with today's hard disk sizes.

Creating the installation

I emulate the university networking setup by setting up a Samba server on my Linux machine. Its clients are virtual

machines.

Samba has been set up for roaming profiles. There is a share for the profiles, an X:-share for home directories and a Z:-share with applications, in exactly the same layout as the university.

I install T_EX Live into the right position on the Z:-share by running the installer on my own Linux system. I select binaries for both Linux and Windows.

I switch between this and my regular installation simply by changing environment variables, for which I have a small shell function. This lets me do much testing and all maintenance from Linux. I explained already that configuration files don't depend on the location of the installation. So it doesn't matter that, seen from Linux, the installation is in a totally different place than it is as seen from Windows.

I populate the texmf-local directory tree with the university house style and some legacy packages. It was almost a straight copy of the corresponding local tree from the previous MiK_TE_X-based installation. For students, there is no need for a local tree.

The 2009 installer

The network installer doesn't have to do much: there are hardly any options, it doesn't have to download and install packages, it just has to add T_EX Live to the search path, create some shortcuts, register an uninstaller and optionally create some file associations.

For most of this, it can use the library functions of the installer and the T_EX Live Manager, both of which are written in Perl.

The following code adds T_EX Live to the search path and creates some menu shortcuts:

```
#!/usr/bin/env perl
BEGIN {
    require "tlmgr.pl";
}

# Only make user-level changes even if admin
$oopts{'w32mode'} = 'user';

# Note. The action_... functions read
# their arguments from @ARGV.

# Add TeX Live to path
unshift @ARGV, 'add';
action_path();

# create some shortcuts
unshift @ARGV, 'install', 'shortcut',
'dviout.win32', 'texworks', 'texlive-en',
'tlpsv.win32';
```

```
action_postaction();
```

File associations can be done similarly. A corresponding uninstaller script:

```
BEGIN {
    require "tlmgr.pl";
}
$oopts{'w32mode'} = 'user';

# remove shortcuts
unshift @ARGV, 'remove', 'shortcut',
'dviout.win32', 'texworks', 'texlive-en',
'tlpsv.win32';
action_postaction();

# Remove TeX Live from path
unshift @ARGV, 'remove';
action_path();
```

Registering and unregistering the uninstaller. However, it is a bit more complicated to register one's custom uninstaller. The T_EX Live modules in tlpkg/TeXLive, and the modules they load, contain everything needed, but the interface is comparatively low-level. Here is the code, for what it is worth:

```
# don't need to re-require modules but
# do need to re-import names
Win32::TieRegistry->import(qw($Registry));
$Registry->Delimiter('/');
$Registry->ArrayValues(0);
$Registry->FixSzNulls(1);

# register uninstaller. Failure not fatal.
my $Master_bsl = $Master;
$Master_bsl =~ s/,/,\\,g;

my $rootkey = $Registry -> Open("CUser",
    {Access =>
        Win32::TieRegistry::KEY_ALL_ACCESS});
my $k;
if ($rootkey) {
    $k = $rootkey->CreateKey(
        "software/microsoft/windows/" .
        "currentversion/uninstall/OurTeXLive/");
    if ($k) {
        $k->{"/DisplayName"} = "OurTeXLive 2009";
        $k->{"/UninstallString"} =
            "\\$Master_bsl\\w32unclient.bat\"";
        $k->{"/DisplayVersion"} = "2009";
        $k->{"/URLInfoAbout"} =
            "http://ourwebsite.edu/ourtexlive";
    }
}
```

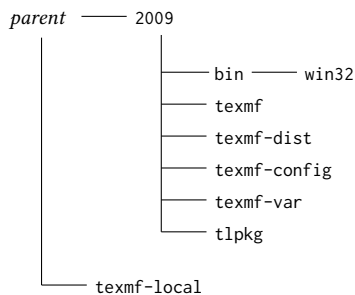
```
warn "Failed to register uninstaller\n"
  unless $k;
and for unregistering the uninstaller:
my $rootkey = $Registry -> Open("User",
  {Access =>
    Win32::TieRegistry::KEY_ALL_ACCESS});
if ($rootkey) { # otherwise fail silently
  my $k = $rootkey->Open(
    "software/microsoft/windows/" .
    "currentversion/uninstall/");
  TeXLive::TLWinGoo::reg_delete_recurse($k,
    'OurTeXLive/') if $k;
}
```

Prototype installer scripts are available at <http://tug.org/texlive/w32client.html>.

ZENWorks. Novell has a tool ZENworks for repackaging applications for NAL. It tracks changes in the registry and the filesystem during installation. The ZEN-generated installer of the repackaged application duplicates those changes. However, for me it was more practical to use a Perl script, and I am grateful to the ICT people that they let me.

Directory layout

We assume the standard T_EX Live directory layout, with texmf-local one level higher than the other trees:



It is possible to choose another layout, e.g. without the year level, but then the components of T_EX Live need a bit more help to find their support files.

Batch wrappers

We also need a wrapper batchfile to make sure that the Perl from T_EX Live is used rather than a locally installed Perl, and to take care that tlmgr.pl, the T_EX Live Manager Perl script, is found. This file is used as a library by our custom installer. Below is a bare-bones wrapper batchfile; in the standard T_EX Live we use much more involved wrappers, with various safeguards.³

```
set this=%~dp0
rem Use TL Perl
```

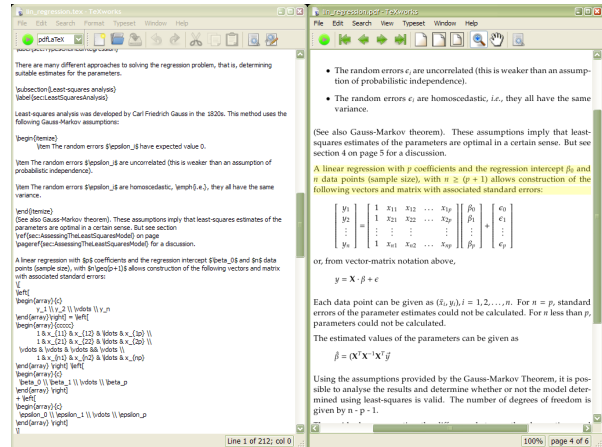


Figure 4. TeXworks for Windows is included in T_EX Live

```
path %this%tlpkg\perl\bin;%this%bin\win32;
  %path%
rem (one line)
set PERL5LIB=%this%tlpkg\perl\lib;
  %this%tlpkg;%this%texmf\scripts\texlive
rem (one line)
```

rem Start Perl script of the same name

```
perl "%~dpn0" %*
```

```
rem Give user opportunity to scan output msgs
pause
```

Note the first line, where the batchfile finds its own directory: set this=%~dp0. This syntax has been available since Windows NT.

The w32client and w32unclient scripts assume that they are in the root of the installation, i.e. in <parent>/2009. However, this is easy to change.

Getting T_EX Live on the network

The first step was asking the ICT department for a directory to install into, preferably with write access for me, so that I can do maintenance without having to involve ICT every time.

The next step was copying everything to that directory. For transport, I nowadays make a giant zip of the installation and put it on a USB stick.

Finally, the installer is integrated into the Novell NAL menu; see figure 3 on page 87. This is done by ICT staff.

The installer places the T_EX Live menu itself under Start / Programs, just as the native installer does.

For maintenance, I used to do small changes manually and do larger changes by wholesale replacement. For the future, rsync looks like a better way.

Additional software

T_EX Live by itself is pretty complete. For Windows, it includes the T_EXworks cross-platform editor (figure 4), and the ps_View PostScript viewer, which can also read pdf. As mentioned earlier, it also contains private copies of Unix mainstays such as Perl and Ghostscript that many T_EX Live components depend upon.

Nevertheless, some additions would be desirable: another editor besides T_EXworks, more graphics support, maybe a bibliography editor.

But there are requirements for such add-ons:

- Free (as in beer)
- Per-user configuration
- Usable for non-geeks

Several programs looked interesting, but didn't meet these requirements or had other problems. They include alternative L^AT_EX editors T_EXmaker and WinShell, bibliography editors JabRef (Java-based) and BibEdt, and a couple of draw programs, IPE and TpX. So I followed the example of previous T_EX Live DVDs and put installers for them in a support subdirectory. Frankly, I don't know whether anybody has made use of these installers.

Editor. For 2008, I decided to stick with T_EXnicCenter as editor. I wrote some fairly elaborate code to configure it for T_EX Live, since the automatic configuration didn't work as nicely for T_EX Live as it did for MiK_TE_X. I also looked at T_EXmaker. It would have been much easier to configure, but at that time it still lacked some important features.

For the 2009 release I'll keep T_EXnicCenter, if only because many users dislike change, but I'll also include T_EXworks, which is already part of standard T_EX Live.

Documentation. The T_EX Live menu contains various shortcuts to manuals, such as the UK FAQ and the 'not so short introduction'. There are also links to the CTAN catalogue and to my own web page for this installation, <http://tex.aanhet.net/miktex/> (!).

Vista and Windows 7

Strictly speaking, this topic doesn't belong here: the network installation only targets XP machines. However, for the standard T_EX Live we had to make sure it would work properly with Vista and Windows 7. Testing this, we ran into some interesting problems.

UAC. Vista introduced User Account Control, or UAC in short. This means, among other things, that only administrators are allowed to install software under Program Files, and only administrators are allowed to change the more important system settings in the registry.

A new slightly annoying twist is that even administrators don't have these privileges automatically. To start a program with admin privileges, you can right-click the shortcut, which usually gives you an option 'Run as administrator'. An administrator has to confirm his intentions, a non-administrator has to provide administrator credentials at this point.

Virtualization. But there is another, more insidious twist: Vista/Win7 may guess that a program needs administrative privileges, e.g. because it has 'install' or 'setup' in its name. If such a program wasn't started with administrative privileges, Vista may fake them. In particular, attempts to write to Program Files might result in writings to *user\appdata\local\virtualstore*. For registry access, similar virtualization might be applied.⁴

Installing T_EX Live with real admin privileges and adding packages with faked admin privileges is not healthy for a T_EX Live installation.

This compatibility mode can be avoided by the addition of a *manifest* which is a bit of XML that explicitly tells Windows under which privileges the program needs to be run. The options are (explanations literally taken from msdn.microsoft.com):

asInvoker The application runs with the same access token as the parent process.

highestAvailable The application runs with the highest privileges the current user can obtain.

requireAdministrator The application runs only for administrators and requires that the application be launched with the full access token of an administrator.

This XML can be embedded into the binary or added as a separate file with the same name as the program, but with *.manifest* appended.

This is our manifest file for the Windows Perl executable:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
  manifestVersion="1.0">
  <assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="*"
    name="perl.exe"
    type="win32"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level="asInvoker"/>
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

We believe that, with the addition of a couple of manifest

files and some tests on admin privileges, T_EX Live 2009 has become Vista-safe.

Conclusion

So you see that maintaining a T_EX installation has little to do with T_EX, slightly more with programming, but is mostly a matter of tying disparate pieces together.

Notes

1. MAPS 33 pp. 59–64 and TUGboat 27:1 pp. 22–27.
2. MiK_TE_X 2.8 may be easier to deal with, but I didn't check this out.
3. T_EX Live even uses a second, binary wrapper around the batch wrapper because some programs handle batchfiles badly.
4. This only happens on 32-bit Vista/Win7.

Siep Kroonenberg
n.s.kroonenberg@rug.nl

Using T_EX's language within a course about functional programming

Abstract

We are in charge of a teaching unit, entitled *Advanced Functional Programming*, for 4th-year university students in Computer Science. This unit is optional within the curriculum, so students attending it are especially interested in programming. The main language studied in this unit is Scheme, but an important part is devoted to general features, e.g., lexical vs. dynamic scoping, limited vs. unlimited extent, call by value vs. call by name or need, etc. As an alternative to other programming languages, T_EX allows us to show a language where dynamic and lexical scoping—`\def` vs. `\edef`—coexist. In addition, we can show how dynamic scoping allows users to customise T_EX's behaviour. Other commands related to strategies are shown, too, e.g., `\expandafter`, `\noexpand`. More generally, T_EX commands are related to macros in more classical programming languages, and we can both emphasise difficulty related to macros and show non-artificial examples. So T_EX is not our unit's main focus, but provides significant help to illustrate some difficult notions.

Keywords

Functional programming, T_EX programming, lexical vs. dynamic scope, macros, evaluation strategies.

Introduction

If we consider *programming* in T_EX [17], we have to admit that this language is old-fashioned, and programs are often viewed as rebuses, as shown in the *Pearls of T_EX programming* demonstrated at BachoT_EX conferences.¹ Some interesting applications exemplifying this language can be found in [19, 30], but as noticed in [5], 'some of these programming tricks are ingenious and even elegant. However [...] it is time for a change'.

So, at first glance, it may be strange to use some examples of T_EX programming within a present-day course devoted to *Functional programming*. Let us recall that this programming paradigm treats computation as the evaluation of mathematical functions, avoiding state and mutable data as much as possible. Functional programming emphasises functions' application, whereas *imper-*

ative programming—the paradigm implemented within more 'traditional' languages, such as C [16]—emphasises changes in state. Many universities include courses about functional programming, examples being reported in [35]. Besides, such languages are sometimes taught as *first* programming languages, according to an approach comparable to [1, 8, 32] in the case of Scheme.

Let us remark that some tools developed as part of T_EX's galaxy have already met functional programming: `cl-bibtex` [18], an extension of B_IB_TE_X—the bibliography processor [26] usually associated with the \mathcal{L} T_EX word processor [20]—is written using ANSI² COMMON LISP [7]; `xindy`, a multilingual index processor for documents written using \mathcal{L} T_EX [24, § 11.3] is based on COMMON LISP, too; B_IB_TE_X2HTML [4], a converter from .bib format—used by the bibliography database files of B_IB_TE_X—to HTML,³ is written in CAML⁴ [21]; M_IB_IB_TE_X,⁵ a re-implementation of B_IB_TE_X focusing on multilingual features [11] is written in Scheme [15]; as another example, Haskell⁶ [28] has been used in [38]; last but not at least, there were proposals for developing $\mathcal{N}\mathcal{J}\mathcal{S}^7$ —a re-implementation of T_EX—using CLOS,⁸ an object-oriented system based on COMMON LISP [39].

The unit we mentioned above is entitled *Advanced Functional Programming*.⁹ It is an optional unit for 4th-year university students in Computer Science, part of the curriculum proposed at the University of Franche-Comté, at the Faculty of Science and Technics, located at Besançon, in the east of France. Most of these students already know a functional programming language: Scheme, because they attended a unit introducing this language in the 2nd academic year in Computer Science.¹⁰ Other students, who attended the first two university years at Belfort, know CAML. So this unit is not an introductory course; we delve thoroughly into functional programming.

In the next section, we expose the 'philosophy' of our unit. Then we summarise the features of T_EX that are useful within this unit and discuss our choice of T_EX. Reading this article only requires basic knowledge of programming; readers who would like to go thoroughly

```
(define (factorial x)
  ;; Returns x! if x is a natural number, the 'false'
  ;; value otherwise.
  (and (integer? x) (not (negative? x))
    (let tr-fact ((counter x)
                  (acc 1))
      ;; Returns acc * counter!.
      (if (zero? counter)
          acc
          (tr-fact (- counter 1)
                    (* acc counter)))))))
```

Figure 1. The factorial function, written using Scheme.

into Scheme constructs we have used throughout our examples can refer to [32], very didactic. Of course, the indisputable reference about T_EX commands is [17].

Our unit's purpose

Functional programming languages have a common root in the λ -calculus, a formal system developed in the 1930s by Alonzo Church to investigate function definition, function application, and recursion [3]. However, these programming languages are very diverse, some—e.g., the Lisp dialects¹¹—are dynamically typed,¹² some—e.g., Standard ML¹³ [27], CAML, Haskell—are strongly typed¹⁴ and include a *type inference mechanism*: end-users do not have to make precise the types of the variables they use, they are inferred by the type-checker; in practice, end-users have to conceive a program using a strongly typed approach because if the type-checker does not succeed in associating a type with an expression, this expression is proclaimed incorrect. As examples, Fig. 1 (resp. 2) show how to program the factorial function in Scheme (resp. COMMON LISP). In both cases, the factorial function we give can be applied to any value, but returns the factorial of this value only if it is a non-negative integer, otherwise, the result is the 'false' value. Fig. 3 gives the same function in Standard ML: it can only be applied to an integer, as reported by the type-checker (see the line beginning with '>').

A course explaining the general principles of functional programming with an overview of some existing functional programming languages would be indigestible for most students, since they could only with difficulty become familiar with several languages, due to the amount of time that can be allocated to each unit. In addition, theoretical notions without practice would not be very useful. So, our unit's first part is devoted to the λ -calculus' bases [10]. Then, all the practical exercises are performed with only one language, Scheme, which most students already

```
(defun factorial (x)
  "Behaves like the namesake function in Scheme
  (cf. Fig. 1)."
  (and
    (integerp x) (not (minusp x))
    (labels ((tr-fact (counter acc)
              ;; The labels special form of
              ;; COMMON LISP introduces local
              ;; recursive functions [33, § 7.5].
              (if (zerop counter)
                  acc
                  (tr-fact (- counter 1)
                          (* acc counter))))))
      (tr-fact x 1))))
```

Figure 2. The factorial function in COMMON LISP.

know. Besides, this unit ends with some advanced features of this language: delayed evaluation, continuations, hygienic macros [9]. In addition, this choice allows us to perform a demonstration of DSSSL¹⁵ [13], initially designed as the stylesheet language for SGML¹⁶ texts. These students attended a unit about XML and XSLT¹⁷ [36] the year before, and DSSSL—which may be viewed as XSLT's ancestor—is based on a subset of Scheme, enriched by specialised libraries.

When we begin to program, the language we are learning is always shown as *finite product*. It has precise rules, precise semantics, and is *consistent*. According to the language used, some applications may be easy or difficult to implement. When you put down a statement, running it often results in something predictable. That hides an important point: a language results from some important choices: does it use lexical or dynamic scoping, or both? To illustrate this notion with some examples in T_EX, that is the difference between the commands `\firstquestion` and `\secondquestion` in Fig. 4. The former can be related to *lexical* scoping, because it uses the value associated with the `\state` command at definition-time and produces:

You're happy, ain't U?

whereas the latter can be related to *dynamic* scoping, because it uses the value of the `\state` command at runtime and yields:

You're afraid, ain't U?

Students find this notion difficult: some know that they can redefine a variable by means of a `let` form in Emacs Lisp [22], but they do not realise that this would be impossible within lexically-scoped languages such as C or Scheme. In other words, they do not have *transversal* culture concerning programming languages,

```

fun factorial x =
  (* If x is a negative integer, the predefined
     exception Domain is raised [27, §§ 4.5–4.7]. The
     internal function tr_fact is defined by means of
     pattern matching [27, § 4.4].
  *)
  if x < 0 then raise Domain
  else let fun tr_fact 0 acc = acc |
            tr_fact counter acc =
              tr_fact (counter - 1)
                acc * counter
          in tr_fact x 1
        end ;
  > val factorial = fn : int -> int

```

Figure 3. The factorial function in Standard ML.

they see each of them as an independent cell, a kind of *black box*.

The central part of our unit aims to emphasise these choices: what are the consequences of a lexical (resp. dynamic) scope? If the language is lexical (resp. dynamic), what kinds of applications are easier to be implemented? Likewise, what are the advantages and drawbacks of the call-by-value¹⁸ strategy vs. call-by-name? In the language you are using, what is variables' extent?¹⁹ Of course, all the answers depend on the programming languages considered. But our point of view is that a course based on Scheme and using other examples in T_EX may be of interest.

T_EX's features shown

As mentioned above, `\def` and `\edef` allow us to illustrate the difference between lexical and dynamic scope. Most present-day programming languages are lexical, but we can observe that the dynamic scoping allows most T_EX commands to be redefined by end-users. The dynamic scope is known to cause *variable captures*,²⁰ but T_EX is protected against undesirable redefinitions by its internal commands, whose names contains the 'e' character. Of course, forcing these internal commands' redefinition is allowed by the `\makeatletter` command, and restoring T_EX's original behaviour is done by the `\makeatother` command.

If we are interested in implementation considerations, the commands within an `\edef`'s body are expanded, so this body is evaluated as far as possible.²¹ To show this point, we can get dynamic scope with an `\edef` command by preventing command expansion by means of `\noexpand`:

```
\edef\thirdquestion{%
```

```

\def\state{happy}
\edef\firstquestion{You're \state, ain't U?\par}
\def\secondquestion{You're \state, ain't U?\par}
\def\state{afraid}

```

Figure 4. Lexical and dynamic scope within T_EX.

```

{\def\firsttwodigits{20}
 \def\lasttwodigits{09}
 \global\edef\thisyear{%
 \firsttwodigits\lasttwodigits}}

```

Figure 5. Using TeX's `\global` command.

```
You're \noexpand\state, ain't U?\par}
and this command \thirdquestion behaves exactly like
\secondquestion (cf. Fig. 4).
```

A second construct, useful for a point of view related to conception, is `\global`, shown in Fig. 5, because it allows 'global' commands to be defined within local environments. There is an equivalent method in Scheme, but not naturally: see Appendix. Let us go on with this figure; any T_EXnician knows that `\thisyear` no longer works if '`\edef`' is replaced by '`\def`'. This illustrates that T_EX commands have limited extent.

Nuances related to notion of equality exist in T_EX: let `\a` be a command already defined:

```
\let\b\a
\def\c{\a}
```

the former expresses that `\a` and `\b` are 'physically' equal, it allows us to retain `\a`'s definition, even it is changed afterwards; the latter expresses an equality at run-time, ensuring that the commands `\c` and `\a` are identical, even if `\a` changes.²²

Scheme's standard does not allow end-users to know whether or not a variable `x` is bound.²³ A T_EXnician would use:

```
\expandafter\ifx\csname x\endcsname\relax...%
 \else...%
\fi
```

For beginners in programming with T_EX, this is quite a complicated statement requiring the commands `\relax` and `\ifx` to be introduced. However, that leads us to introduce not only the construct `\csname... \endcsname`, but also `\expandafter`, which may be viewed as kind of call by value. A simpler example of using this strategy is given by:

```
\uppercase\expandafter{\romannumeral 2009}
```

—which yields 'MMIX'—since this predefined command `\uppercase` is given its only argument as it is; so

putting the `\expandafter` command causes this argument to be expanded, whereas removing it would produce 'mmix', because `\uppercase` would leave the group `{\romannumeral 2009}` untouched, then `\romannumeral` would just be applied to 2009. That is, T_EX commands are *macros*²⁴ in the sense of 'more classical' programming languages.

The last feature we are concerned with is *mixfixed* terms, related to parsing problems and priorities. T_EX can put mixfixed terms into action by means of *delimiters* in a command's argument, as in:

```
\def\put(#1,#2)#3{...}
```

Discussion

As shown in the previous section, we use T_EX as a 'cultural complement' for alternative constructs and implementations. Sometimes, we explain differences by historical considerations: for example, the difference between `\def` and `\long\def`—that is, the difference in L^AT_EX between `\textbf{...}` and `\begin{bfseries}... \end{bfseries}`—comes from performance considerations, since at the time T_EX came out, computers were not as efficient as today. Nevertheless, are there other languages that could be successfully used as support of our unit? Yes and no.

An interesting example could be COMMON LISP. Nevertheless, this language is less used now than some years ago, and it is complexified by the use of several *namespaces*.²⁵ Besides, this language's initial library is as big as possible; it uses old constructs.²⁶ That is why we give some examples in COMMON LISP, but prefer for our course to be based on Scheme, which is 'the' modern Lisp dialect, from our point of view.

Concerning the coexistence of lexical and dynamic variables, the Perl²⁷ language [37] provides it. In addition, it has been successfully used to develop large software packages, so examples could be credible. However, it seems to us that dynamic variables in Perl are rarely used in practice. In fact, the two dynamic languages mainly used are Emacs Lisp and T_EX, in the sense that end-users may perceive this point. From our point of view, using examples in Emacs Lisp requires good knowledge about the emacs²⁸ editor, whereas we can isolate, among T_EX's features, the parts that suit us, omitting additional details about T_EX's tasks. Likewise, such an approach would be more difficult with Perl.

Conclusion

Our unit is viewed as theoretical, whereas other optional units are more practical, so only a few students attend ours. But in general, students who choose it do not regret it, and in fact enjoy it. They say that they have clear ideas about programming after attending it. Some students view our examples in T_EX as a historical curiosity since this language is quite old and originates from the 1980s, but they are surprised by its expressive power. Some, that are interested in using T_EX more intensively, can connect programming in T_EX to concepts present in more modern languages.

Acknowledgements

When I decided to use T_EX to demonstrate 'alternative' implementations of some features related to programming, I was quite doubtful about the result, even if I knew that some were interested. But feedback was positive, some students were encouraged to go thoroughly into implementing new T_EX commands for their reports and asked me for some questions about that. Thanks to them, they encouraged me to go on with this way in turn.

Appendix: `\global` in Scheme

In this appendix, we show that a construct like `\global` in T_EX may be needed. Then we will explain why it cannot be implemented in Scheme in a 'natural' way.

Let us consider that we are handling *dimensions*, that is, a number and a measurement unit—given as a symbol—like in T_EX or DSSSL. A robust solution consists of using a list prefixed by a *marker*—e.g., `((*dimension*) 1609344 mm)`—such that all the lists representing dimensions—of type *dimension*—share the same head element, defined once. To put this marker only when the components—a number and a unit²⁹—are well-formed, it is better for the access to this marker to be restricted to the functions interfacing this structure. So here is a proposal for an implementation of functions dealing with dimensions:

```
(let ((*marker* '(*dimension*)))
  (define (mk-dimension r unit)
    ;; Performs some checks and creates an
    ;; object of type dimension, whose
    ;; components are r and unit.
    . . . )
  (define (dimension? x)
    ;; Returns #t if x is of type dimension, #f
    ;; otherwise.
    . . . )
  (define (dimension->mm dimension-0)
```

```

(define mk-dimension)
(define dimension?)
(define dimension->mm)

(let ((*marker* '(*dimension*)) ; Only the cell's address is relevant.
      (allowed-unit-alist
        '((cm . ,(lambda (r) (* 10 r))) ; Each recognised unit is associated with a function giving the
          (mm . ,values)))) ; corresponding length in millimeters.
  (set! mk-dimension
        (let ((allowed-units (map car allowed-unit-alist))
              (lambda (r unit)
                (and (real? r) (>= r 0) (memq unit allowed-units) (list *marker* r unit))))))
  (set! dimension? (lambda (x) (and (pair? x) (eq? (car x) *marker*))))
  (set! dimension->mm
        (lambda (dimension-0) ; dimension-0 is supposed to be of type dimension.
          ((cdr (assq (caddr dimension-0) allowed-unit-alist)) (cadr dimension-0)))))

```

Figure 6. Global definitions sharing a common lexical environment in Scheme.

```

;; Returns the value of dimension-0,
;; expressed in millimeters.
. . . )

```

Unfortunately, this does not work, because `define` special forms inside the scope of a `let` special form are viewed as *local* definitions, like `\def` inside a group in \TeX . So, `mk-dimension`, `dimension?`, and `dimension->mm` become inaccessible as soon as this `let` form is processed. The solution is to define these three variables globally, and modify them inside a local environment, as shown in Fig. 6.

This *modus operandi* is quite artificial, because it uses side effects, whereas functional programming aims to avoid such as far as possible. But in reality, from a point of view related to conception, there is no ‘actual’ side effect, in the sense that variables like `mk-dimension`, `dimension?`, and `dimension->mm` would have been first given values, and then modified. The first bindings may be viewed as *preliminary declarations*,³⁰ however, using ‘global’ declarations for variables introduced within a local environment would be clearer, as in \TeX . To sum up, such an example illustrates that some use of assignment forms are not related to actual side effects, and \TeX ’s `\global` command allows us to explain how this example could appear using a ‘more functional’ form, without any side effect.³¹

References

- [1] Harold ABELSON and Gerald Jay SUSSMAN, with Julie SUSSMAN: *Structure and Interpretation of Computer Programs*. The MIT Press, McGraw-Hill Book Company. 1985.
- [2] Neil BRADLEY: *The Concise SGML Companion*. Addison-Wesley. 1997.
- [3] Alonzo CHURCH: *The Calculi of Lambda-Conversion*. Princeton University Press. 1941.
- [4] Jean-Christophe FILLIÂTRE and Claude MARCHÉ: *The Bib \TeX 2HTML Home Page*. June 2006. <http://www.lri.fr/~filliatr/bibtex2html/>.
- [5] Jonathan FINE: “ \TeX as a Callable Function”. In: *Euro \TeX 2002*, pp. 26–30. Bachotek, Poland. April 2002.
- [6] Michael J. GORDON, Arthur J. MILNER and Christopher P. WADSWORTH: *Edinburgh LCF*. No. 78 in LNCS. Springer-Verlag. 1979.
- [7] Paul GRAHAM: *ANSI COMMON LISP*. Series in Artificial Intelligence. Prentice Hall, Englewood Cliffs, New Jersey. 1996.
- [8] Jean-Michel HUFFLEN : *Programmation fonctionnelle en Scheme. De la conception à la mise en œuvre*. Masson. Mars 1996.
- [9] Jean-Michel HUFFLEN : *Programmation fonctionnelle avancée. Notes de cours et exercices*. Polycopié. Besançon. Juillet 1997.
- [10] Jean-Michel HUFFLEN : *Introduction au λ -calcul (version révisée et étendue)*. Polycopié. Besançon. Février 1998.
- [11] Jean-Michel HUFFLEN: “A Tour around ML $\text{Bib}\TeX$ and Its Implementation(s)”. *Biuletyn GUST*, Vol. 20, pp. 21–28. In *Bacho \TeX 2004 conference*. April 2004.
- [12] Jean-Michel HUFFLEN: “Managing Languages within ML $\text{Bib}\TeX$ ”. *TUGboat*, Vol. 30, no. 1, pp. 49–57. July 2009.
- [13] International Standard ISO/IEC 10179:1996(E): DSSSL. 1996.

- [14] *Java Technology*. March 2008. <http://java.sun.com>.
- [15] Richard KELSEY, William D. CLINGER, Jonathan A. REES, Harold ABELSON, Norman I. ADAMS IV, David H. BARTLEY, Gary BROOKS, R. Kent DYBVIK, Daniel P. FRIEDMAN, Robert HALSTEAD, Chris HANSON, Christopher T. HAYNES, Eugene Edmund KOHLBECKER, JR, Donald OXLEY, Kent M. PITMAN, Guillermo J. ROZAS, Guy Lewis STEELE, JR, Gerald Jay SUSSMAN and Mitchell WAND: "Revised⁵ Report on the Algorithmic Language Scheme". HOSC, Vol. 11, no. 1, pp. 7–105. August 1998.
- [16] Brian W. KERNIGHAN and Dennis M. RITCHIE: *The C Programming Language*. 2nd edition. Prentice Hall. 1988.
- [17] Donald Ervin KNUTH: *Computers & Typesetting, Vol. A: The T_EXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.
- [18] Matthias KÖPPE: *A BibT_EX System in Common Lisp*. January 2003. <http://www.nongnu.org/cl-bibtex>.
- [19] Thomas LACHAND-ROBERT : *La maîtrise de T_EX et L^AT_EX*. Masson. 1995.
- [20] Leslie LAMPORT: *L^AT_EX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.
- [21] Xavier LEROY, Damien DOLIGEZ, Jacques GARRIGUE, Didier RÉMY and Jérôme VOUILLOIN: *The Objective Caml System. Release 0.9. Documentation and User's Manual*. 2004. <http://caml.inria.fr/pub/docs/manual-ocaml/index.html>.
- [22] Bill LEWIS, Dan LALIBERTE, Richard M. STALLMAN and THE GNU MANUAL GROUP: *GNU Emacs Lisp Reference Manual for Emacs Version 21. Revision 2.8*. January 2002. <http://www.gnu.org/software/emacs/elisp-manual/>.
- [23] John MCCARTHY: "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". *Communications of the ACM*, Vol. 3, no. 4, pp. 184–195. April 1960.
- [24] Frank MITTELBACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L^AT_EX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [25] Chuck MUSCIANO and Bill KENNEDY: *HTML & XHTML: The Definitive Guide*. 5th edition. O'Reilly & Associates, Inc. August 2002.
- [26] Oren PATASHNIK: *BibT_EXing*. February 1988. Part of the BibT_EX distribution.
- [27] Lawrence C. PAULSON: *ML for the Working Programmer*. 2nd edition. Cambridge University Press. 1996.
- [28] Simon PEYTON JONES, ed.: *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press. April 2003.
- [29] Erik T. RAY: *Learning XML*. O'Reilly & Associates, Inc. January 2001.
- [30] Denis B. ROEGEL : « Anatomie d'une macro » . *Cahiers GUTenberg*, Vol. 31, p. 19–27. Décembre 1998.
- [31] Michael SPERBER, William CLINGER, R. Kent DYBVIK, Matthew FLATT, Anton VAN STRAATEN, Richard KELSEY, Jonathan REES, Robert Bruce FINDLER and Jacob MATTHEWS: *Revised⁶ Report on the Algorithmic Language Scheme*. September 2007. <http://www.r6rs.org>.
- [32] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.
- [33] Guy Lewis STEELE, JR., with Scott E. FAHLMAN, Richard P. GABRIEL, David A. MOON, Daniel L. WEINREB, Daniel Gureasko BOBROW, Linda G. DEMICHEL, Sonya E. KEENE, Gregor KICZALES, Crispin PERDUE, Kent M. PITMAN, Richard WATERS and Jon L WHITE: *COMMON LISP. The Language. Second Edition*. Digital Press. 1990.
- [34] "T_EX Beauties and Oddities. A Permanent Call for T_EX Pearls". In *T_EX: at a turning point, or at the crossroads? BachoT_EX 2009*, pp. 59–65. April 2009.
- [35] Simon THOMPSON and Steve HILL: "Functional Programming through the Curriculum". In: *FPLE '95*, pp. 85–102. Nijmegen, The Netherlands. December 1995.
- [36] W3C: *XSL Transformations (XSLT). Version 2.0*. W3C Recommendation. Edited by Michael H. Kay. January 2007. <http://www.w3.org/TR/2007/WD-xslt20-20070123>.
- [37] Larry WALL, Tom CHRISTIANSEN and Jon ORWANT: *Programming Perl*. 3rd edition. O'Reilly & Associates, Inc. July 2000.
- [38] Halina WĄTRÓBSKA i Ryszard KUBIAK: „Od XML-a do T_EX-a, używając Emacsa i Haskella”. *Biuletyn GUST*, tom 23, strony 35–39. In *BachoT_EX 2006 conference*. kweiecień 2006.
- [39] Jiří ZLATUŠKA: "λS: Programming Languages and Paradigms". In: *EuroT_EX 1999*, pp. 241–245. Heidelberg (Germany). September 1999.

Notes

1. The most recent pearls can be found in [34].
2. American National Standards Institute.
3. HyperText Markup Language, the language of Web pages. [25] is a good introduction to it.
4. Categorical Abstract Machine Language.
5. MultiLingual Bib_TE_X.
6. This language was named after logician Haskell Brooks Curry (1900–1982).
7. New Typesetting System. It was finally developed using Java [14].
8. COMMON LISP Object System.
9. ‘*Programmation fonctionnelle avancée*’ in French. In 2009, it has been renamed to ‘*Outils pour le développement*’ (*Tools for Development*), but changes in the contents are slight.
10. The program of this 2nd academic year unit can be found in French in [8].
11. ‘Lisp’ stands for ‘LIST Processing, because Lisp dialects’ major structure is linked lists. Their syntax is common and based on fully-parenthesised prefixed expressions. Lisp’s first version, designed by John McCarthy, came out in 1958 [23]. This language has many descendants, the most used nowadays being COMMON LISP and Scheme.
12. ‘Dynamically typed’ means that we can know the type of an object at run-time. Examples are given in Figs. 1 & 2.
13. ‘ML’ stands for ‘MetaLanguage’ and has been initially developed within the formal proof system LCF (Logic for Computable Functions) [6]. Later on, it appears as an actual programming language, usable outside this system, and its standardisation resulted in the Standard ML language.
14. There are several definitions of *strong typing*. The most used within functional programming is that the variables are typed at compile-time. Some courses at the same level are based on a strongly typed functional programming language, examples being CAML or Haskell. Is that choice better than Scheme? This is a lively debate... but it is certain that these courses do not emphasise the same notions as a course based on a Lisp dialect.
15. Document Style Semantics Specification Language.
16. Standard Generalised Markup Language. Ancestor of XML (eXtensible Markup Language); it is only of historical interest now. Readers interested in SGML (resp. XML) can refer to [2] (resp. [29]).
17. eXtensible Stylesheet Language Transformations.
18. Nowadays, the call by value is the most commonly used strategy—in particular, in C and in Scheme—the argument expression(s) of a function are evaluated before applying the function. For example, the evaluation of the expression (factorial (+ 1 9))—see Fig. 1—begins with evaluating (+ 9 1) into 10, and then factorial is applied to 10. In other strategies, such as *call by name* or *call by need*, argument expressions are evaluated whilst the function is applied.
19. The *extent* of a variable may be viewed as its lifetime: if it is *limited*, the variable disappears as soon as the execution of the block establishing it terminates; if it is *unlimited*, the variable exists as long as the possibility of reference remains. In Scheme, variables have unlimited extent.
20. A *variable capture* occurs when a binding other than the expected one is used.
21. On the contrary, Scheme interpreters do not evaluate a

lambda expression’s body. They use a technique—so-called *lexical closure*—allowing the function to retrieve its definition environment.

22. There is another distinction in Scheme, between ‘physical’ (function eq?) and ‘visual’ equality (function equal?) [31, § 11.5].
23. COMMON LISP allows that about variables and functions, by means of the functions boundp and fboundp [33, 7.1.1].
24. Macros exist in Scheme: the best way to implement them is the use of *hygienic* macros, working by pattern-matching [31, §§ 11.2.2 & 11.19].
25. As an example of handling several namespaces in COMMON LISP, let is used for local variables, whereas local recursive functions are introduced by labels, as shown in Fig. 2.
26. For example, there is no hygienic macro in COMMON LISP.
27. Practical Extraction Report Language.
28. Editing MACros.
29. ... although we consider only centimeters and millimeters in the example given in Fig. 6, for sake of simplicity.
30. In Scheme’s last version, a variable can be defined without an associated value [31, § 11.2.1]. That was not the case in the version before [15, § 5.2], so such a variable declaration was given a *dummy* value, which enforced the use of side effects.
31. Many data structures, comparable with our type *dimension*, are used within ML_{B_TE_X}’s implementation, as briefly sketched in [12]. Another technique, based on *message-passing*, allows us to avoid side effects. Only one function would be defined to manage dimensions, and the three functionalities implemented by mk-dimension, dimension?, and dimension->mm would be implemented by messages sent to the general function, the result being itself a function.

Jean-Michel Hufflen
LIFC (EA CNRS 4157),
University of Franche-Comté, 16, route de Gray,
25030 Besançon Cedex, France

Introducing new French-speaking users to \LaTeX quickly and convincingly

Abstract

For four university years, we had to introduce 2nd-year university students in Mathematics to \LaTeX . An important goal was to make them able to use \LaTeX when they are given some long homework in Mathematics the year after (3rd-year university). This teaching unit only included lab classes and was 15 hours long. We present our approach in detail and explain how it was perceived by students.

Keywords

Teaching \LaTeX , successive steps of a course, lab-class-based curriculum, students' perception

Introduction

When \LaTeX [23] came out, it was sometimes viewed as a program hard to use, except for computer scientists familiar with hermetic programming languages. However this word processor has become more and more well-known as a powerful tool that produces high-quality print output. Besides, beginners can learn it now with many books introducing it, in many languages, some—non-limitative—examples are [13] in English, [4, 22, 30] in French, [27] in German, [5, 34] in Hungarian, [3] in Italian, [7] in Polish, [31] in modern Greek, ... In addition, some universities propose introductions to \LaTeX within their *curricula*. An example is a unit—entitled *Scientific Tools*—we taught for four academic years (2004–2008), at the Faculty of Science and Technics, part of the University of Franche-Comté and located at Besançon, in the east of France.

Students who attended this unit were in the 2nd academic year of Mathematics.¹ One goal of this teaching unit was to ease the writing of an important homework the year after, that is, within the 3rd academic year in Mathematics, so a substantial part of this unit was devoted to \LaTeX 's math mode. Let us be precise that this teaching unit was not optional; that is, all the students had to attend it, even if they were not convinced of \LaTeX *a priori*. Of course, some had heard about it, some had not. This unit only included lab classes and was 15 hours long. So students actually practised exercises in \LaTeX ,

but we did not have enough time to show very advanced features.

We think that the approach we follow is interesting. In a first section, we make explicit our requirements and the pitfalls we wanted to avoid. Then we show the broad outlines of the steps of our unit and summarise the experience we got. Of course, reading this article only requires basic knowledge about \LaTeX .

What to do? What to avoid?

Many introductions to \LaTeX begin with typing a small text and enriching it; some examples are [5, 22]. Our starting point is that this *modus operandi* has too many drawbacks, especially for non-English-speaking future users, in particular for French-speaking ones. First, only a few students are familiar with typing texts quickly and intensively, even if some have already used computers. They may make some typing mistakes in command names. Of course, any \LaTeX teacher is able to fix them, but the price to pay is loss of time and dynamic. Besides, students need to be convinced of \LaTeX from their first experiments. They should see that this word processor is suitable for large-sized texts, at the beginning, they should be able to observe that it is easy with \LaTeX to apply some changes related to layout: changing characters' basic size, switching one-column and two-column layouts, ... All these goals can be reached only if students are given a text already typed and ready to be processed. That is, compiling this text should be successful the first time, so there is no anxiety about this point.

Besides, let us not forget that the most natural choice for a text to be typed by French students is a text in French. But some typographical rules are different from English ones: for example, a thin space—produced by the \LaTeX command ‘\,’—is to be put just before a ‘high’ punctuation sign,² such as an exclamation or question mark:³

Joie, bonheur et délectation !

whereas the same signs are glued to the preceding text in English:

there was a lot of fun!

That is, such punctuation signs should be *active*⁴ within French fragments. Of course, the simplest solution to this problem is to use the babel package's french option [25, Ch. 9]. So, end-users can type 'Vous_comprenez?' or 'Vous_comprenez_?' and the results will be typeset correctly in both cases:

Vous comprenez ?

This point may seem to be a digression, but our purpose is to show how difficult the beginning of an introduction to \LaTeX for non-English-speaking people is. Teachers are placed in a dilemma: either students have to typeset texts peppered with commands such as '\ ' or '\,', or they should be given a big preamble, consisting of many \usepackage directives, with the advice 'You will understand later.'⁵ In the case of the French language, this point is enforced because of accented letters: the most frequently used are directly provided by French keyboards—for example, 'é' or 'è', very frequent within French words—but the keys are unusable if the inputenc package has not been loaded with the latin1 option [25, § 7.11.3]. If French students begin to learn \LaTeX by typing their own texts, there is no doubt that these texts will contain accented letters.

Anyway, the best solution seems to be a complete text—in French or in English—and students can perform first exercises by changing some sentences or adding some short fragments. Students can put down some simple sentences in English, so writing in this language avoids some problems related to French typography. When they have become familiar with the commands of \LaTeX and its 'philosophy', the tools making it practical to write in the French language—the babel and inputenc packages—will be introduced. From our point of view, a 'good' text, usable as a starting point, should provide the following features:

- a title, author, and date identified, so students can learn about commands such as \title, \author, \date, and \maketitle; an annotation may be a pretext for introducing the \thanks command;
- an average-sized text;
- a command used without argument, in order to show that a space character following a command's name without explicit delimiter is gobbled up;
- a word hyphenated incorrectly, so students can realise that some words may be hyphenated, and learn how to fix such a mistake, even if that is rare;
- a pretext for introducing a new command in \LaTeX ,
- a pretext for introducing cross-references.

The steps of our unit

The source text given to students is [10]. More precisely, the first version, giavitto.tex, does not use the babel package, even though this text is in French, with a short introduction we wrote in English. The inputenc package is not used, either, so we used \TeX accent commands, and 'high' punctuation signs are explicitly preceded by a thin space, e.g.:

Joie, bonheur et d\ 'e}lection\,!

This text, a criticism about a book, came out in a forum. It has seemed to us to be very suitable for such an introduction to \LaTeX , because:

- it is 3 pages long, that is, a small-sized text, but not too short;
- the introduction's second paragraph reads:

... using the \LaTeX\ word processor...

- without the babel package's french option, there is a word hyphenated between a consonant and the following vowel, which is incorrect in French:⁶ 'ex-emple' (for 'example', hyphenated as 'ex-ample' in English) should be hyphenated as 'exem-ple';⁷
- in this text, some words need an emphasis stronger than what is usually expressed by italicised characters: in the original text, typeset using only the standard typewriter font,⁸ these words were written using capital letters:

Comment pouvait-IL savoir cela ?

The source text reads:

Comment pouvait-\superemph{il}...

and we can illustrate the use of variants of this new command \superemph:

```
\newcommand{\superemph}[1]{\uppercase{#1}}
\newcommand{\superemph}[1]{%
  **\uppercase{#1}**}
\newcommand{\superemph}[1]{**\textsc{#1}**}
...
```

That allows a kind of 'semantic markup' [17], in the sense that this markup is related to a semantic notion, rather than some layout.

The first exercise is to compile the source text of this first version giavitto.tex, the second is to play with some options of the \documentclass command: twocolumn,

12pt, ... so students can see that adapting a document to different layouts is easy. Guidelines are given in [15]. To sum up the order we follow:

- basic notions: commands, environments, preamble;
- sectioning commands: `\part`, `\chapter`, ...
- parsing problem regarding commands without a right delimiter (*cf. supra*);
- formatting environments: `center`, `flushleft`, `flushright`;
- changing characters' look: commands and environments such as `\textbf` and `bfseries`, `\textsf` and `sffamily`, ...
- introducing and redefining new commands: `\newcommand` and `\renewcommand`, use of 'semantic' markup, by means of commands such as `\superemph` (*cf. supra*), 'local' definitions—surrounded by additional braces—vs. global ones;
- changing size: commands and environments `small`, `footnotesize`, ...
- list environments: `itemize`, `description`, `enumerate`; counters controlling `enumerate` environments, difference between redefining *values* and *look*—e.g., as done respectively by the commands `\enumi` and `\labelenumi`—insertion of footnotes;
- introducing *packages*: examples are `indentfirst`⁹ [25, p. 32] and `eurosym` [25, pp. 408–409];
- notion of *dimensions*, how the page layout parameters are defined [25, Fig. 4.1] and how to customise them;
- how sequence of words (resp. successive lines) are split into lines (resp. pages), useful commands such as `\-`, `\linebreak`, `\pagebreak`, `\smallskip`, `\medskip`, `\bigskip`, putting unbreakable space characters by means of the '~' input character;
- management of *cross-references* and introduction of *auxiliary* (.aux) files, commands `\label`, `\ref`, `\pageref`; use of an additional .toc file for a table of contents and `\tableofcontents` command;
- introducing some basic differences between French and English typography; then we show how the `babel` package allows L^AT_EX to typeset texts written in many languages, possibly within the same document; introducing some useful commands of the `babel` package's french option; the 'standard' preamble of a L^AT_EX document written in French is given:

```
\documentclass{...}
```

```
\usepackage[...]{babel}
```

```
\usepackage[T1]{fontenc}
```

```
\usepackage[latin1]{inputenc}
```

```
...
```

(see [25, §§ 7.11.3 & 7.11.4] about the packages `fontenc`

and `inputenc`); as an example taking advantage of L^AT_EX's multilingual features as much as possible, a second version of [10], `giavitto-plus.tex`, is given to students;

- the document's end is devoted to some complements not demonstrated in lab classes: some converters to HTML¹⁰ (L^AT_EX2HTML [11, Ch. 3], T_EX4ht [11, Ch. 4], HyperL^AT_EX [19]), BibL^AT_EX [28].

Of course, students are not required to master all these items: we make precise the points students must know, and other information is given as a *memorandum*, e.g., the list of commands changing characters' look. A second document [16] is devoted to math mode and is organised as follows:

- math mode vs. text mode;
- spacing in math mode;
- commands changing characters' look in math mode, e.g., `\mathrm`, `\mathit`, ..., additional packages such as `amssymb` or `euscript`;¹¹
- commands producing Greek letters for mathematical purpose (`\alpha`, ...) and symbols (`\leftarrow`, ...) in math mode;
- subscripts, superscripts, fractions, radicals;
- adjustments: commands `\displaystyle`, `\textstyle`, ..., operators *taking limits* or not, horizontal and vertical *struts*, the `amsmath` package's `\text` command;
- definition of operator names, by means of the commands `\mathop`, `\DeclareMathOperator`, `\DeclareMathOperator*`, `\mathbin`, `\mathrel`;
- *delimiter* management, by means of the commands `\left`, `\middle`, and `\right`;
- environments cases and equation; more features belonging to the `amsmath` package, such as the environments `multline`, `split`, `gather`, and the commands `\tag`, `\intertext`;
- environments belonging to L^AT_EX: `eqnarray`, `eqnarray*`;
- environments useful for general matrices (`[b|p|v|V]matrix`) and arrays (`[sub]array`), packages `multirow`, `[del]array`;
- back to L^AT_EX's text mode and introduction of the tabular environment.

Two other documents gently bring this unit to its end:

- [29] introduces `pdfLATEX` and the `hyperref` package [11, Ch. 2], taking as much advantage as possible of the features of the PDF¹² format related to hyperlinks;

- [1] is devoted to image insertion, by means of the packages `graphic(s|x)` [25, § 10.2].

Of course, all these documents include references—possibly on-line—that allow readers to learn more.

Lessons learned

Teaching this unit gave good results: it actually seemed to us that students really enjoy discovering \LaTeX and using it. Generally they got nice outputs. In addition, practising \LaTeX 's math mode caused them to realise how diverse 'graphical' expressions of Mathematics are. For example, 'modulo' is both an infix and prefixed operator, as reflected by the two commands `\bmod` and `\pmod`. Likewise, the notion of operators taking limits separates the layout—the location of each piece of information—and the common notion—an interval's endpoints. That is, the commands of \LaTeX 's math mode may be viewed as *presentation markup*, comparable to the namesake notion in MathML¹³ [33, § 2.1.2].

Anyway, let us recall that we taught students in Mathematics. Such students learned the basics of a programming language,¹⁴ but do not plan to become computer scientists. So, they did not get used to presenting programs nicely, by indenting them, as students in Computer Science learn to do, in order to be able to work on them again. A good exercise to emphasise this point is first to give students a complicated formula to be typeset, then to ask them to change it.

Teachers have to give some advice about organising \LaTeX source texts. For example, there should be no other word on a line containing `\begin` or `\end` commands delimiting environments, and nesting environments should be made clear by indenting them:

```
... text before.
\begin{itemize}
  \item ...
  \begin{itemize}
    \item ...
    ...
  \end{itemize}
\end{itemize}
Text after...
```

Some notations should be avoided when a more systematic markup is available. For example, we think that it is better for students to get used to writing:

```
\begin{small}
...
\end{small}
```

than `{\small ...}`. Of course, the latter may appear

as simpler for short fragments, but any \TeX nician knows that it is possible to use a command like `\small` without additional braces, in which case, this size change runs until the next size change command. If the markup associated with a command is not clearly expressed, some students may be baffled. Besides, let us consider three versions of an end-user defined command typesetting a note using small-sized characters:

```
\newcommand{\note}[1]{%
  \begin{small}#1\end{small}}
\newcommand{\noteone}[1]{\small#1} % Wrong!
\newcommand{\notetwo}[1]{\{\small#1}}
```

Of course, any \LaTeX teacher can explain why the `\noteone` command does not work as expected, and how to fix this wrong definition as done for the `\notetwo` command. However, a user who is used to `small` as an environment—rather than `{\small ...}`—would probably put down this `\note` command as we did, and that is indisputably the simplest solution.

The commands and environments introduced by the \LaTeX format have homogeneous taxonomy about delimiting arguments and effects. That is, the markup is very clear, in particular for beginners. That may not be the case for commands originating from \TeX 's basis: for example, if you would like to put a vertical strut whose length is given, we can use the construct `\vbox to 1.1\baselineskip{}` [16, § 2.7], that is, using a kind of *mixed* markup. Of course, dealing with such a command is rare. But other commands belonging to *plain \TeX* 's math mode, such as `\over` or `\atop`, are error-prone since they have no argument and are only related to positional relationships. Let us compare plain \TeX 's `\over` command with \LaTeX 's `\frac`, that has 2 arguments: the source texts for the numerator and denominator.¹⁵

Last but not least, we notice some points related to the implementation we used: `\TeXnicCenter` [32], built on top of the `MiK \TeX` typesetting engine [24], and running under the Windows operating system. This graphic interface is very good, especially the correspondence between the editor's cursor and a marker in the resulting `.dvi`¹⁶ file. The main drawback is that `MiK \TeX` runs in non-stop mode. As a consequence, students may get almost complete texts in case of recoverable errors. So they do not have to be aware of their errors and they perceive only 'serious' ones. It is needed to introduce them to `.log` files, and ask them to tolerate only warning messages.

Conclusion

\LaTeX being extensible because of its numerous packages, it is impossible for an introductory course to give all the

functionalities that already exist. In fact, teachers also have to show how to use \LaTeX 's documentation—good documents exist in French—to learn more on their own. But it is essential for students to understand \LaTeX 's philosophy and get good methods. We think our method fulfills these goals. From 2003 to 2005, J.-M. Hufflen taught 4th-year university students enrolled in 'Digital Publishing' program¹⁷ at the Letter Faculty of Besançon, and got initial experiences for writing [15]. A more concise document [2] has been used by A.-M. Aebischer for analogous introductory courses given at the IREM¹⁸ institute for future teachers in Mathematics.

Acknowledgements

Sincere thanks to Karl Berry and Barbara Beeton, who proofread the first version of this article.

References

- [1] Anne-Marie AEBISCHER : *Insertion d'images*. Document de support de travaux pratiques. Avril 2008.
- [2] Anne-Marie AEBISCHER : *Créer des documents scientifiques avec \LaTeX* . Stage 2008 à l'IREM de Franche-Comté. 2008.
- [3] Claudio BECCARI : *Introduzione all'arte della composizione tipografica con \LaTeX* . September 2009. GUIT.
- [4] Denis BITOUZÉ et Jean-Côme CHARPENTIER : *\LaTeX* . Pearson Education. 2006.
- [5] BUJDOSÓ Gyöngyi – FAZEKAS Attila: *\TeX kezdépek*. Tertia Kiadó, Budapest. április 1997.
- [6] *The Chicago Manual of Style*. The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.
- [7] Antoni DILLER: *\LaTeX wiersz po wierszu*. Wydawnictwo Helio, Gliwice. Polish translation of *\LaTeX Line by Line* with an additional annex by Jan Jelowicki. 2001.
- [8] Bernard GAULLE : *Notice d'utilisation de l'extension frenchpro pour \LaTeX . Version V5,995*. Avril 2005. <http://www.frenchpro6.com/frenchpro/french/ALIRE.pdf>.
- [9] Bernard GAULLE : *L'extension frenchle pour \LaTeX . Notice d'utilisation. Version V5,9993*. Février 2007. <http://www.tug.org/texlive/Contents/live/texmf-dist/doc/latex/frenchle/frenchle.pdf>.
- [10] Jean-Louis GIAVITTO : *Ouverture des veines et autres distractions*. Documents de support de travaux pratiques. <http://lifc.univ-fcomte.fr/home/~jmhufflen/l2s/giavitto.pdf> et <http://lifc.univ-fcomte.fr/home/~jmhufflen/l2s/giavitto-plus.pdf>. Octobre 1986.
- [11] Michel GOOSSENS and Sebastian RAHTZ, with Eitan M. GURARI, Ross MOORE and Robert S. SUTOR: *The \LaTeX Web Companion*. Addison-Wesley Longman, Inc., Reading, Massachusetts. May 1999.
- [12] Maurice GRÉVISSE : *Le bon usage*. Duculot. Grammaire française. 12^e édition refondue par André Goosse. 1988.
- [13] David GRIFFITHS and Desmond HIGHAM: *Learning \LaTeX* . SIAM. 1997.
- [14] Jean-Michel HUFFLEN : « Typographie : les conventions, la tradition, les goûts, ... et \LaTeX ». *Cahiers GUTenberg*, Vol. 35–36, p. 169–214. In *Actes du congrès GUTenberg 2000*, Toulouse. Mai 2000.
- [15] Jean-Michel HUFFLEN : *Premier contact avec \LaTeX* . Document de support de travaux pratiques. <http://lifc.univ-fcomte.fr/home/~jmhufflen/l2s/to-do.pdf>. Janvier 2006.
- [16] Jean-Michel HUFFLEN : *Mode mathématique*. Document de support de travaux pratiques. <http://lifc.univ-fcomte.fr/home/~jmhufflen/l2s/to-do-math.pdf>. Février 2006.
- [17] Jean-Michel HUFFLEN: "Writing Structured and Semantics-Oriented Documents: \TeX vs. XML". *Biuletyn GUST*, Vol. 23, pp. 104–108. In *Bacho \TeX 2006 conference*. April 2006.
- [18] Jean-Michel HUFFLEN : *C++... et d'autres outils... pour l'étudiant mathématicien*. Polycopié. Besançon. Janvier 2008.
- [19] *Hyper \LaTeX* . February 2004. <http://hyperlatex.sourceforge.net>.
- [20] *Java Technology*. March 2008. <http://java.sun.com>.
- [21] Donald Ervin KNUTH: *Computers & Typesetting. Vol. A: The \TeX book*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.
- [22] Thomas LACHAND-ROBERT : *La maîtrise de \TeX et \LaTeX* . Masson. 1995.
- [23] Leslie LAMPOR: *\LaTeX : A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.
- [24] *MiK \TeX ... Typesetting Beautiful Documents*. 2009. <http://miktex.org/>.
- [25] Frank MITTELBACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The*

- LaTeX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [26] Chuck MUSCIANO and Bill KENNEDY: *HTML: The Definitive Guide*. 6th edition. O'Reilly & Associates, Inc. October 2006.
- [27] Elke NIEDERMAIR und Michael NIEDERMAIR: *LaTeX—Das Praxisbuch*. 3. Auflage. Franzis. 2006.
- [28] Oren PATASHNIK: *BibTeXing*. February 1988. Part of the BibTeX distribution.
- [29] François PÉTIARD : *Le package hyperref*. Document de travaux pratiques. Mai 2005.
- [30] Christian ROLLAND : *LaTeX par la pratique*. O'Reilly France, Paris. 1999.
- [31] Apostolos SYROPOULOS: *LaTeX*. Ένας Πληρης για την Εκμαθηση του Συστηματος Στοιχειοθεσιας LaTeX. Παρατηρητης. 1998.
- [32] *TeXnicCenter*. 2008. <http://www.texniccenter.org/>.
- [33] W3C: *Mathematical Markup Language (MathML) Version 2.0*, 2nd edition. W3C Recommendation. Edited by David CARLISLE, Patrick ION, Robert MINER, and Nico POPPELIER. October 2003. <http://www.w3.org/TR/2003/REC-MathML2-20031021>.
- [34] WETTL Ferenc – MAYER Gyula – SZABÓ Péter: *LaTeX kézikönyv*. Panem. 2004.
8. Let us recall that this text came out in October 1986; the interfaces used within these forums were not comparable to the Web.
9. Indenting the first paragraph after a display heading is more customary in French text than in English, so introducing this indentfirst package is relevant in our unit.
10. HyperText Markup Language, the language of Web pages. [26] is a good introduction to it.
11. Most of the math mode's advanced features are described in detail in [25, Ch. 8].
12. Portable Document Format, Adobe's format.
13. MATHematical Markup Language [33] is an XML (eXtensible Markup Language) application for describing mathematical notation regarding either its structure or its content. Let us mention that MathML's broad outlines are taught to 5th-year students in Statistical Modelling ('*Master 2 de Mathématiques, mention Modélisation statistique*', in French) [18, ch. 9], as part of a unit entitled 'Software Engineering'.
14. Java [20], in the *curricula* of the Faculty of Science and Technics located at Besançon.
15. As mentioned in Note 13, there is an introduction to MathML for some 5th-year university students in Mathematics. MathML's content model [33, § 2.1.3], more related to the semantics of mathematical expressions, is easier to understand for these students than the presentation model.
16. DeVice-Independent.
17. '*Master 1 d'Édition numérique*', in French.
18. '*Institut de Recherche sur l'Enseignement des Mathématiques*', that is, 'Research Institute about teaching Mathematics'.

Notes

1. '*License 2, parcours Mathématiques et Mathématiques appliquées*', w.r.t. French terminology.
2. This notion of 'high' sign of punctuation belongs to French typography's terminology. A short survey of these rules is given in [6, §§ 9.21–9.33], a more complete reference is [14], in French.
3. The following quotations come from [10].
4. This notion is explained in [21, Ch. 7].
5. So do [4, 7, 27, 31] The first example of [34] does not use any package, the `\usepackage` command being introduced immediately after. In addition, examples given at first are small-sized, so introducing some variants—e.g., `twocolumn` vs. `onecolumn`—would not be very convincing. On another subject, French texts can be typeset using the packages `french(pro)le` [8, 9], as alternatives to the `babel` package, but the problem of introducing such a package at the course's beginning remains the same.
6. Except for etymological hyphenation, now hardly used in practice.
7. In fact, this point is debatable, because some French typography manuals consider that a word should not be hyphenated before a silent syllable—'*exemple*' sounds as [ɛgzɑ̃pl]. (That is why this word is not hyphenated in the version processed with the `babel` package's `french` option.) But these typography manuals mention that this convention is difficult to follow, in particular when text columns are narrow, as in daily newspapers, for example. More details about this point can be found in [12].

Anne-Marie Aebischer¹

Bruno Aebischer¹

Jean-Michel Hufflen²

François Pétiard¹

¹ Department of Mathematics (UMR CNRS 6623),
University of Franche-Comté, 16, route de Gray,
25030 Besançon Cedex, France.

² LIFC (EA CNRS 4157),
University of Franche-Comté, 16, route de Gray,
25030 Besançon Cedex, France.

Oriental T_EX by a dummy

Abstract

This article is converted from the slides presented at the conference.

What is Oriental T_EX

- It is a project by Idris Samawi Hamid, Taco Hoekwater and Hans Hagen.
- The project started shortly after we started the LuaT_EX project.
- It boosted development of LuaT_EX thanks to a grant that paid for coding LuaT_EX.
- It also boosted the development of ConT_EXt MkIV and was a real good torture test for OpenType font support.
- This project also costs us a whole lot of time.
- The main objective is to let T_EX typeset high quality (traditional) Arabic.
- Closely related to this is to extend ConT_EXt capabilities to deal with advanced critical editions.
- In the meantime a high quality Arabic OpenType font has become part of the package.

How we proceed

- Of course we were a bit too optimistic when setting the time schedule for this project.
- This is because we need to have quite some bits and pieces in place beforehand.
- For instance, making the font and perfecting OpenType support involves a lot of trial and error and testing.
- This is mostly due to lack of specifications, benchmarks and limitations in tools.
- We have identified the needs for critical editions but have postponed some of that till we have opened up more of LuaT_EX.
- We are also getting a better picture of what is needed for advanced right-to-left typesetting, especially in mixed directionality.

Simple OpenType fonts

- In Latin scripts we have mostly one-to-one and many-to-one substitutions.
- This can happen in sequence (multiple passes).
- Sometimes surrounding characters (or shapes) play a role.
- In some cases glyphs have to be (re)positioned relative to each other.
- Often the substitution logic is flawed and it is assumed that features are applied selectively (DTP: select and apply).
- Of course this is unacceptable for what we have in mind.

The Oriental T_EX approach

- We put as much logic in the font as possible, but also provide a dedicated paragraph builder (written in Lua).
- The so-called First-Order Analysis puts a given character into isolated, initial, middle, or final state.
- The Second-order Analysis looks at the characters and relates this state to what characters precede or succeed it.

- Based on that state we do character substitutions. There can be multiple analysis and replacements in sequence.
- We can do some simple aesthetic stretching and additional related replacements.
- We need to attach identity marks and vowels in proper but nice looking places.
- In most cases we're then done. Contrary to other fonts we don't use many ligatures but compose characters.

But we go further

- The previous steps already give reasonable results and implementing it also nicely went along with the development of Lua \TeX and Con \TeX t MkIV.
- Currently we're working on extending and perfecting the font to support what we call Third-Order Contextual Analysis.
- This boils down to an interplay between the paragraph builder and additional font features.
- In order to get pleasing spacing we apply further substitutions, this time with wider or narrower shapes.
- When this is done we need to reattach identity marks and vowels.
- Optionally we can apply HZ-like stretching as a finishing touch.

Look at luatex

- no order (kh ī t ā w [u] l)
- first order
- second order
- second order (Jeem-stacking)
- minimal stretching
- maximal stretching (level 3)
- chopped letter khaa (for e.g. underlining)

Hans Hagen
Pragma ADE, Hasselt

(kheetawul)

ل و ا ت ي خ
ل و ا ت ي خ
ل و ا ت ي خ
ل و ا ت ي خ
ل و ا ت ي خ
ل و ا ت ي خ
ل و ا ت ي خ

Writing Pitman shorthand with Metafont and L^AT_EX

Abstract

With pen shorthand, the traditional speech-recording method, unwritten speech is at first manually captured and then transliterated into a digital text. We have built programs which reverse the second step of this process, i.e. transform text into shorthand.

Here we present as a special case an online system, which converts English text into Pitman 2000 shorthand using Metafont and L^AT_EX. The impact of our system on pattern recognition of handwritten shorthand and on stenography teaching is discussed.

In order to approximate the speed of speech, alphabet based shorthand systems make use of phonetic writing, abbreviations and simplified writing, thus reducing the redundancy of the orthographic code and the graphic redundancy of longhand characters.

In the following sections we exemplify these principles with the *Pitman shorthand language* (abbreviated as PSL) and describe how the Pitman 2000 shorthand system can be implemented in Metafont [4].

Elements of PSL

A glyph of one or more words as denoted with PSL, the so-called **stenem** is composed of

- an outline** consisting of joined consonant signs, written without lifting the pen from the paper, and
- diacritics** corresponding to vowel phonemes.

The stenem components are written in this order.

An example: The stenem of the word 'rote' \nearrow , pronounced as r * ou t is built of the outline \nearrow , formed by joining the strokes (r)=/ and (t)=|, the signs¹ of the consonant phonemes r and t and the heavy dash sign [ou], the diacritical mark of the vowel ou, following /.

The signs of consonant phonemes. These signs, also called **strokes**, are either line segments or quarter circles:

\backslash	\backslash			//	—	—	
(p)	(b)	(t)	(d)	(ch)	(jh)	(k)	(g)
(f)	(v)	(th)	(dh)	(s)	(z)	(sh)	(zh)
⌒	⌒	(())	⌒	⌒

Though invented in 1837, the PSL design is guided by modern phonological classifications and principles [7, 8].

Thus the signs of voiced consonants are more firmly written variants of their unvoiced counterparts. Friction vs. occlusion of a consonant is denoted by rounding the corresponding sign (cf. the rows). A change of the place of articulation causes a change of slant in consonant signs (cf. the columns).

Remaining strokes² are:

nasals				liquids		
⌒	⌒	⌒		⌒	/	⌒
(m)	(n)	(ng)		(l)	(r)	(_r)

The signs \frown \smile \smile $_$ $_$ are horizontals, \frown \smile are upstrokes,³ all other consonant signs are downstrokes.

Vowel, diphthong and triphone signs. These diacritical signs are placed alongside a consonant sign, before or after it, depending on whether the vowel is read before or after the consonant, i.e. going from the beginning of a stroke on the left-hand or the right-hand side of upstrokes and horizontals if the vowel is read before or after the consonant. Places are changed for downstrokes.

before $\overset{2}{\curvearrowright} \overset{3}{\curvearrowright}$ $\overset{1}{\curvearrowright}$ after 'ell' 'lay' 'us' 'so'

Twelve **vowel diacritics** are realized in PSL. They are differentiated by their glyph (light or heavy, dot or dash) and its position. Any consonant sign has three places for a vowel sign to be located according to the direction in which the consonant stroke is written: at the beginning (1st), in the middle (2nd) or at the end (3rd place).

place				
1 st	[a]	[ah]	[o]	[oo]
		↘		⌋
	'at'	'pa'	'odd'	'saw'
2 nd	[e]	[ei]	[uh]	[ou]
			↘	⌋
	'ed'	'aid'	'up'	'no'
3 rd	[i]	[ii]	[u]	[uu]
	⌈	⌈	⌊	⌊
	'ill'	'eel'	'took'	'coup'

It can be seen from this table that the light vowel signs are reserved for the short vowels and are put in the same places as the heavy vowel signs for the long vowels.

The table proceeds row-wise (over the position) from signs for opened vowels to signs for closed vowels and column-wise from dots for front vowels to dashes⁴) for back vowels. Compare a such as in 'at', which is an opened front vowel with the closed back vowel uu, such as the one in 'coup' at the opposite vertices of the table.⁵

There are four **diphthong signs** at 1st and 3rd places:

place		
1 st	[ai] 'my'	[oi] 'joy'
	⌈	⌈
3 rd	[ow] 'out'	[yuu] 'few'
	⌋	⌋

The **triphone signs**, indicated by a small tick attached to a diphthong sign, represent any vowel following the diphthong, as in:

⌈ 'diary', ⌈ 'loyal', ⌈ 'towel' and ⌈ 'fewer'.

There is also a special diphone sign for other vowel combinations put in the place of the first vowel. Consider the second mark for the diphone ia in 'idea' ⌈ put at the 3rd place – the place of [i].

Observe also, that the first vowel in a word decides where the first upstroke or downstroke of the outline will be written – above, on or through the line.

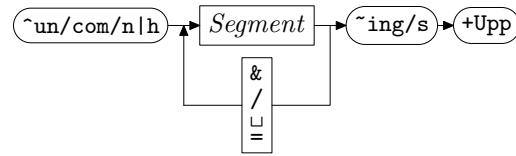
↘	↘	↘
'pa'	'pay'	'pea'

Stenems

We propose here a complete PSL grammar and describe it by means of syntax diagrams.⁶ A terminal result-

ing from grammar productions is called **metaform**, e.g. (r) [ou]&(t) is the metaform⁷ corresponding to ⌈, the stemem of the word 'rote'.

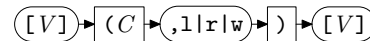
Stenems are composed of segments with (mostly) joined (&-Notation) outline.



Stenems can start with a morphological prefix such as ~com, ~con, ... and they can end with a verbal suffix, such as ~ing. Both are realized by a mark, such as a light dot, before the first and/or after the last segment outline, respectively. Last suffix +Upp indicates proper names, whose glyphs are underlined in PSL.

⌈ ~com [o] (n) (g) [ou] ~ing [a] (n) +Upp

Segments. The core of a segment



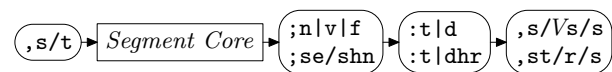
is built of an obligatory consonant sign (C) framed by optional vowel signs [V]. The strokes of Section 1 can be modified to express the frequent case of a consonant followed by an r or an l – written by an initial right or left hook,⁸ respectively:

(p,r) [ei] (f,r) [ii] (p,l) [ei] (f,l) [ii]

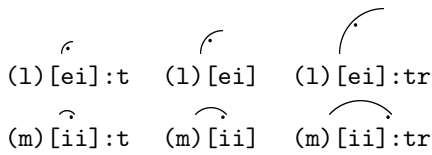
The segments are the counterparts of the syllables, hence there is a provision for vowel signs occurring between two strokes – 1st and 2nd place vowel signs are written after the first stroke whereas the otherwise ambiguous 3rd place vowel signs are written before the second stroke:

⌈ 1 3 | 2 3 | (t) [e]&(r)&[i] (t) [ou]&(r) [i]
t * e . r i . t our . r iy

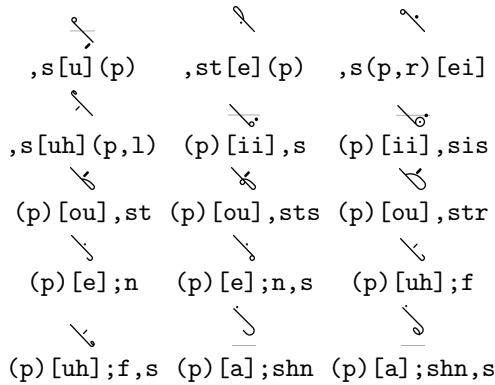
The following syntax diagram completes the definition of a PSL segment



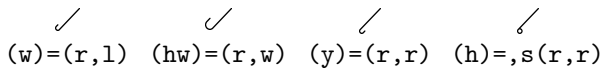
At first PSL strokes come at three sizes – half-sized (suffix :t/d), normal or double-sized (suffix :tr/dr/dhr), e.g.:



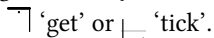
Additionally strokes can be prefixed and/or suffixed by left or right, small or larger hooks, circles, loops and cracknels,⁹ for example:



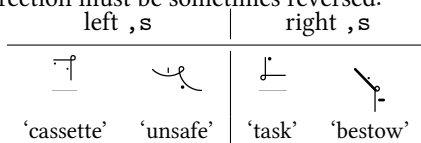
Observe also how the (not previously mentioned) signs of the consonants w, hw, y and y are defined:



Joining and disjoining the segments. Stenem outlines are written mostly without lifting the pen; typically the segments are joined at obtuse outer angles, e.g.:



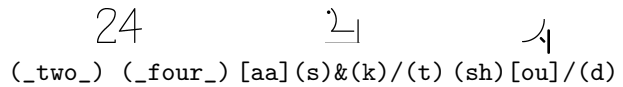
Special care must be taken for ,s-circles. As an ,s-circle is normally written within the curves or as a left circle otherwise, and as writing the circle outside an angle is the simplest way of joining two segments – the circle direction must be sometimes reversed:



However there are singular cases, where a continuous connection is not possible, consider



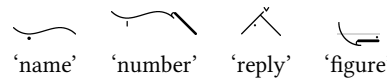
Also when writing numbers or when writing the endings t or d in past tense of regular verbs, the segments are disjoined:¹⁰



Metafont implementation. Elementary strokes (circle arcs or straight lines) and the circles, hooks, ... used for prefixes/suffixes are realized as splines with curvature=0 at both ends. Thus trivially consonantal parts of segments, when joined tangent continuously are curvature continuous, too [5].

Technically speaking the diacritics are an array of discontinuous marker paths, while the outline is an array of (mostly) continuous only¹¹ Metafont-paths.¹²

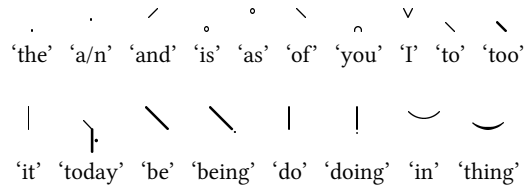
Besides the circled connections also the m/n joinings were made curvature continuous; joinings with cusps exist, too:



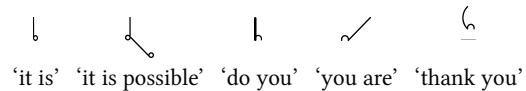
Abbreviations

In PSL, short forms, phrases and intersections are used for frequently occurring words.

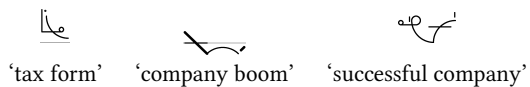
Short forms are either arbitrary strokes or shortened outlines largely without diacritics:



Phrases in PSL are simply stenems of two and more words connected together:¹³



Also one stroke may be struck through a preceding one in commonly used collocations. Examples of such **intersections** are:¹⁴

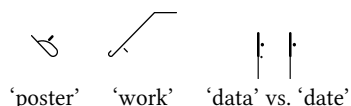


Short forms, phrases and intersections are to be learned by heart. Our system maintains an abbreviation dictionary of (word, metaform) tuples written in `lexc` [1].

text2Pitman

text2Pitman is an online system,¹⁵ which records input text as Pitman 2000 shorthand. Just as in [5, 6] the conversion is done in four steps:

1. The input is tokenized. Tokens with a metaform entry in the abbreviation dictionary are separated from other words.
2. For a word in the latter category we find its pronunciation in Unisyn accent-independent keyword lexicon [2]. The non/writing of minor vowels, the so-called schwas (ə),¹⁶ is guided by special PSL rules: in secondary stress syllables most of them are ignored ('poster'), rhotic schwas are written out ('work') and some others are to be back-transformed¹⁷ ('data' vs. 'date'):



The pronunciation string is then transformed to a metaform by the stenemizer – a program coded as the tokenizer above in the XEROX-FST tool `xfst` [1]. The transformation is carried out by a series of cascaded context dependent rewrite rules, realized by finite state transducers (FSTs). Decomposition of a stenem into its constituent segments as done by the stenemizer is unique, but as the underlying PSL grammar allows ambiguities,¹⁸ the metaform found is not always the one commonly used:¹⁹

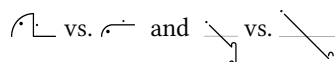
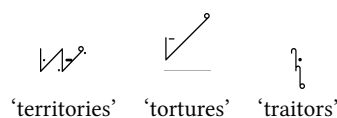


Figure 1.



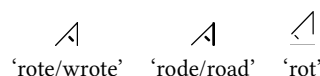
3. In a `mf` run for each of the tokens using the code resulting from its metaform a Metafont character is generated.
4. The text is then set with \LaTeX , rendered with `dvips`, ... and sent as an image to the browser.

Remarks on pattern recognition

`text2Pitman` can provide test samples for the reversed procedure – the pattern recognition of handwritten PSL. This task is done in three major steps:²⁰

1. **Shape recognition yielding the metaform.**
This step requires at first the recognition of the mid points of segments and of the slope as well of the curvature sign there. Then the prefixes and suffixes have to be found and classified.
2. **Conversion of the metaform into pronunciation strings.**
As our stenemizer is a two-level²¹ `xfst` transducer, this could be accomplished by reversing its order of operation, but it is more elaborate. Shorthand writers often omit vowel diacritics in some words, such as:

It is not harmful in long outlines,²² but for short stenems the correct use of diacritics and observing the right overall position is essential. Consider an example of words with nearly the same PSL outline:



As most recognizers do not detect line thickness, still more shorthand homographs result. Thus, this complex task can be handled only by taking into account the word frequencies and using a weighted transducer. Nevertheless our system could automatically create a knowledge base of (metaform, pronunciation string(s)) entries.

3. The transliteration of the pronunciation strings into English with correct orthography is difficult because of the numerous and very frequent English language homophones.²³

Educational uses of the software

A novel dvitype-based DjVu-backend of our software to produce a text-annotated and searchable shorthand record, which can be viewed with a standard DjVu-plugin to a browser or a standalone viewer. Moving with the mouse over a stenem displays the originating word(s), as can be seen in figure 1.

Compare our “live” record with a printed textbook, where the writing or reading “Exercises” are separated from the “Keys to Exercises”.

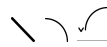
It is probable that as shorthand usage declines, publishers of shorthand books will not, as in the past, insist on their proprietary solutions. In any case, our web server based software suggests a future with centralized dictionaries and textbooks utilized and maintained by an interested user community.

References

- [1] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, Stanford, 2003.
- [2] Susan Fitt. Unisyn multi-accent lexicon, 2006. <http://www.cstr.ed.ac.uk/projects/unisyn/>.
- [3] Swe Myo Htwe, Colin Higgins, Graham Leedham, and Ma Yang. Knowledge based transcription of handwritten Pitman’s Shorthand using word frequency and context. In *7th International Conference on Development and Application Systems*, pages 508–512, Suceava, Romania, 2004.
- [4] Donald E. Knuth. *The Metafontbook*, volume C of *Computers and Typesetting*. Addison-Wesley Publishing Company, Reading, Mass., 5th edition, 1990.
- [5] Stanislav J. Šarman. Writing Gregg Shorthand with Metafont and L^AT_EX. *TUGboat*, 29(3):458–461, 2008. TUG 2008 Conference Proceedings.
- [6] Stanislav J. Šarman. DEK-Verkehrsschrift mit Metafont und L^AT_EX. *Die T_EXnische Komödie*, 21(1):7–20, 2009.
- [7] Bohumil Trnka. *A Phonological Analysis of Present-day Standard English*. Prague, 1935. Revised Edition, 1966.
- [8] Bohumil Trnka. *Pokus o vědeckou teorii a praktickou reformu těsnopisu. An Attempt at the Theory of Shorthand and its Practical Application to the Czech Language*. Facultas Philosophica Universitatis Carolinae Pragensis, Sbirka pojednání a rozprav XX, Prague, 1937.

Notes

1. In the following, phonemes are denoted in typewriter type, the corresponding consonant signs are parenthesized, and vowel, diphthong and triphone signs are bracketed.
2. There are two signs for r. For the signs of h, w, wh and y see Section 2.1.
3. (1) can be written in both directions.
4. Dashes are written at right angles to the stroke at the point where they are placed.
5. which is nearly Jones’ IPA vowel quadrilateral reflected.
6. Optional vs. obligatory parts are enclosed in rounded boxes; nonterminals are written in cursive, terminals in typewriter type.
7. The metaform without intervening non-letters corresponds linearly (stress and schwas excluded), to the pronunciation of a word, e.g. (r) [ou]&(t) ↔ r * ou t
8. ,r is written within the rounded curves while ,l is symbolized by a larger hook.
9. Not all of the $3 \times 2^4 \times 2^4$ thinkable prefix/suffix combinations can actually occur, e.g. at the beginning of English words only the following three consonant sequences spr, str, skr, spl and skw are possible [7]. Segments starting/ending with ,s-circles are very common.
10. then the notation □ or / is used
11. PSL is classified as one of the so-called geometric shorthand systems, which contrast with cursive systems resembling smooth longhand writing.
12. drawn either with thick or thin Metafont pens or filled.
13. The most common “consonant sign” is the word space.
14. With strokes (f) for ‘form’ and (k) for ‘company’, resp.
15. See our project web site, and also DEK.php for the German shorthand DEK and Gregg.php for Gregg shorthand counterparts.
16. the most frequent “(non)vowels”
17. both to their spelling equivalent
18. ‘LaTeX’ as (l) [ei]&(t) [e]&(k) vs. (1) [ei]:t&[e] (k) and ‘computer’ as ^com(p) [yuu]&(t,r) vs. ^com(p) [yuu]:tr. The metaform can be interactively adjusted.
19. here the first variant
20. See [3] and the references there. We comment on these steps using our terminology.
21. Lexical transducers carry out both (e.g. morphological) analysis and synthesis.
22. Although the words shown have the same sequence of consonants, their outlines are distinct.
23. ‘I, eye’, ‘wright, right, rite, write’, ‘hear, here’, ‘by, buy, bye’ are the most frequent.



Stanislav Jan Šarman
 Computing Centre
 Clausthal University of Technology
 Erzstr. 51
 38678 Clausthal
 Germany
 Sarman (at) rz dot tu-clausthal dot de
<http://www3.rz.tu-clausthal.de/~rzsjs/steno/Pitman.php>