✍

TEX Paradigms

TEX inside, insight, in sight: get priorities right

Kees van der Laan

Abstract

It is argued that for using TEX — a multi-level tool — TEX inside knowledge should not be necessary for the layman. The 'why, what and when,' especially at the grey level will be discussed. Necessary insight issues are enumerated.

Introduction

We all love TEX, that is those people who gather at user meetings. But, ... apparently — and the theme of the GUST 97 meeting witnesses it, as was the earlier TUG slogan 'develop your TEXpertise' — there is a need for knowledge of TEX's inner workings. In general I don't agree.[1] Wizards do need it, but what does the layman?

TEX inside, or ... insight in TEX?

Below material has been borrowed from my earlier Paradigm notes, and some more.

Levels of understanding. TEX, and its user interfaces, can be seen as a well-documented multi-level tool for EP, and communication in general. Its quality, general availability, being for free, openness, and extensibility are necessary issues which contributed to its success. Its abstraction from input and output devices are genuine gems.

Are there any disadvantages? For daily use a tool should be simple, should be of the push-the--button type to a high degree. Knuth envisioned even that users should create their own blend of TEX for any significant application, adapted to the purpose and therefore simple in use.[2]

What I did in BLUe TEX was precisely in Knuth's spirit of personalizing TEX, though not at the level Knuth foresaw. I did it on the macro level, by the way like Knuth with his manmac — the macro collection he used for typestting *The TEXbook* and METAFONTbook — and like all other writers of macro collections.[3] I abstracted form the microscopics, from the complexity, in abstracting towards macroscopic commands, in the push-the-button spirit, such that I could forget about most of the details of TEX, except from some general knowledge. Similarly to Knuth, and differently from most other macro writers, I adopted the minimal markup paradigm.[4]

Copy can best be setup just in ASCII with visual layout, and only when finished with the contents a few markup commands need to be inserted, et voilà you will end up with a beautiful typeset result, at the expense of little markup, and as nearly an unblurred script as possible.[5]

Example 1. From the PWT guide

Let us assume that we have ordinary text, to be typeset by computer. What to do? Let us begin by the beginning and write it down.

```
An unexpected party

In a hole in the ground there lived a hobbit.
```

1: It is similar to driving cars. A little nodding knowledge is beneficial. Inside knowledge was only necessary in the beginning.

2: What happened was that TEX has been ported to many if not all conceivable platforms. METAFONT has been adapted for graphical pusposes as extra to font generation in Knuth's spirit by John Hobby in his METAPOST.

3: ϵ-TEX is different. More in the spirit of what Knuth foresaw. It added primitives next to the processing modes: compatible, extendible, enhanceble. However, it is not driven by a concrete, huge, boundary-challenging task. Therefore, I can't tell the difference from 'creeping featurism.' Nothing wrong with that, IMHO, with all respect. It bypasses, however, the personalizing aspect of TEX, that is many a TEXie personalizes TEX in order to fulfill his/her not-so-huge nor boundary-challeging tasks, towards a *simpler language*, abstracting from details, eventually in the jargon of his/her trade. A much overlooked aspect IMHO.

4: By this I mean that no superfluous (markup) symbols should be necessary — as an example think of the curly braces mania — nor should concepts alien to the field (EP) be introduced. It should be natural and behave the way you expect it to behave, in short TEX should be customized.

5: I agree that this is an oversimplification. In reality I directly start from an empty template for my notes.

```
Not a nasty, dirty, wet hole filled with the
ends of worms and an oozy smell,
nor yet a dry, bare sandy hole with nothing
in it to sit down on or to eat:
it was a hobbit-hole and that means comfort.

It had a perfectly round door like a porthole,
painted green,...
```

We must instruct the computer — that is, supply markup — to make something nice out of it. How?

As far as I understand it, Knuth's markup comes down to just preceding the title by `\beginchapter`. That is all. He makes use of the already available visual markup of blank lines. These blank lines separate document parts, be it a title from the text or paragraphs from each other.

Is that all? Essentially, yes. But — there is always a but — the role which `\beginchapter` has to play is not simple. Happily, that is at this moment of use not our concern. That is the job of the book designer in cooperation with the programmer, or if we did it ourselves some activity from the past.

I take for granted the niceties of TEX's automatically text processing features — and which a regular user tends to forget — such as automatic handling of ligatures, justification and hyphenation over paragraphs, and ignoring of superfluous spaces in text. All this is not necessarily restricted to English. Again, nodding knowledge is beneficial.

Only after many a year I realized that the best way of using TEX is not to use it to start with. Just ASCII with visual layout, with as top-priority to get the contents right.

Once we are able to achieve this, we may continue, because it's of no use to have a beautiful typeset document of which the contents is rubbish, trivial, and in general not worth to be communicated.

To make a long story short: I abstracted from the tool to such an extend that I could forget about most of it.

Audience: the user life-cycle. Users are alive and tend to move: they come, develop and go. I presume they will end up as knowledgeable users, who . . . will forget about most of the details of the tool. They have lost interest, are burnt out.

Script: minimal markup. The result of the above attitude is that the markup of the script at the outer level is simple and equivalent to SGML, modula some syntactic sugar. I not only like the typeset results but also the elegance of TEX markup.

Another benefit comes with reuse. My main reuse is to convert my BLUe TEX scripts conforming to MAPS style, that is into LATEX. For the outer level markup I use my convertor-assistant written in TEX, which replace `\bluehead...` into `\section{...}` and so on; no retyping.[6]

Lower level markup tags — (complex) tables, math, graphics and so on — are left invariant and I include the corresponding macros along with the converted script. This way of working is similar to handling PostScript graphics. There too the invokes are left invariant and the converted script is accompanied by the PostScript files.

TEX as a blackbox?

Can TEX be used as such? Alas, history has it, that (computer-assisted) typesetting is too complex a job. We need greybox awareness, at best by customizing TEX first towards your personal needs. IMHO, we need awareness of what is going on and how to discern our wishes into realistic ones.

Example 2. Verbatim in headings

In this (TEXnical) note I needed a control sequence as part of the heading. The backslash was needed. How to do that? At the expense of another font — kind and size — in the heading a non-context sensitive solution is the use of `\cs`.

```
\blueexample \cs{begindisplay}...
```

It appears correctly in the heading and in the ToC. With `\blueexample` and ilks the in-line verbatim does not work, even worse horrible error messages come your way. BLUe is not robust — as is the same with any personalized format, I guess — in the sense that you can use anything anywhere is not yet in sight, alas. Moreover, its size should by default kameleon.

TEX insight: greyboxes

I grew up with the paradigm of 'greyboxes,' especially in numerical mathematics. It was considered

6: I must confess, however, that it has developed little as yet, so it is a personal tool. Of course, one can use Perl or similar scripting tools, but I liked to exercise TEX for this, with as result that it is sufficient for my purpose.

impractical as a user to loose oneself in the details. To understand roughly the ideas underlying the algorithm, its virtues and limitations was enough for using a library routine. Take for example the infamous gem of zerofinding as inplemented in the zerofinder `zeroin`. The complexity of the mixture of strategies is not necessary to be known by the user. For a user the invariance: a shrinking interval with the function values at the endpoints of opposite sign, is all that has to be known. After completion the interval is smaller than the requested precision. In other words a small interval will result on which the function changes sign. For a continuous function this implies that a zero of the function has been located within the required precision. Awareness that convergence is superlinear does not harm.

Why greyboxes in TEX?. Macrowriting — restricted to creating your push-the-button personalized level of TEX — needs greyboxes awareness.

Is it handy to know more, to understand TEX the program?

I don't know, I guess it should be handy, but at what price, what is the investment versus the benefits? I could do without, which saved me a lot of time and ... mental luggage. For that reason I consider myself as not to belong to the TEX-wizards clan, if ever I would like to belong to a clan.

Basically, I needed enough knowledge to be able to read and understand the already available macro collections. And I must say that a few of these collections are just too clever, too complicated and too huge. A lot of dependencies not in the least due to metaness — misplaced IMHO, with all respect — and they suffer from unduly general philosophies, while a pragmatic here-and-now approach would have served the purpose, not in the least the maintenance issue, casu quo the adaptability to change in general.[7]

What greyboxes knowledge of TEX?. Necessary is

- tokens are characterized by ASCII-catcode pair
- I/O abstraction
- the boxes, glue and penalties approach
- linebreaking works over paragraphs (how paragraph parameters work)
- how copy is reused (stored within TEX or within a file)
- awareness of (rigid) discrete sized fonts sets.

For macro writers the following extras

- processing on-the-fly via two-part macros
- the concept of the active list separator
- the use of token variables for options.

Let me give you an example, the macro to typeset a heading, and to reuse the title in the toc and index. Maybe all the above knowledge is not necessary if we start from templates, for example the `\beginsection` macro as supplied by Knuth. The toc and index requirements introduce a lot of extra difficulties. Let us first concentrate on the limited problem and impose ourselves some restrictions. Why not go for simplicity first?

Heading. First of all we should clearly state the design goals, which activity by the way is much neglected in the TEX world. Much is done on-the-fly.

Design goals From a typography point of view I would like for the typesetting of the heading to

- discourage to set the title alone at the end of a page
- typeset (flexible) vertical space before (big) and after (med) the title
- gobble spaces at the beginning of the title
- set the title in bold face (and the current size) unindented
- don't indent the first line of the text after.

Furthermore, I would like to reuse the title in a running head or a ToC.

What are the problems? The markup language constructs have to be chosen. In macro expansion the two-part macro TEXnique should be the basis, IMHO, with all respect. In BLUe's format I adopted the pairs `\begin`⟨*tag*⟩ and `\end`⟨*tag*⟩.

In order to set the title loose from the context we have to determine values for the glue, and be sure that it disappears at the top of a page. The coding solution below has different values for the parameters as supplied in `\beginsection` of *The*

7: My philosophy on coping with change and variety is to separate stable issues from volatile aspects. I found that a User Interface is most susceptable to variety and change, especially when it is not as simple as simple can be, paradoxally as it may seem. For TEX and friends this means that UI's vary and conversions have to be made, where the functionalities of the macroscopics commands vary too, making full-proof automated conversion impossible. Lowerlevel TEXnical issues have been proven rather stable, and could be used in any flavour of TEX.

TeXbook. From experiments it turned out that the values supplied by Knuth don't give nice results with BLUe. Too much glue was inserted for my taste.

Another problem is to prevent the head title to be printed at the bottom of a page on its own. The latter is related to the 'widow-orphan' phenomenon.

The coding of the minimal markup variant without parsing but with processing on-the-fly of 'the argument' has to be resolved in some elegant way.

Required greyboxes knowledge In principle awareness is needed of

- two-part macrowriting
- MVL splitting details[8]

I consider it more pragmatic to start from a template by Knuth — `\beginsection` *The TeXbook* 355 — where the design considerations have been taken into account. I made it a little more modular by starting from two-part macros and by introducing the token variables `\pre`⟨*tag*⟩ and `\post`⟨*tag*⟩, supporting the dissection of concerns adago. They parameterize the placement of a document element within context. Whether you talk about titles, displays, tables, graphs, or you name it. Perhaps this should be generalized in something like `\pasteup`⟨*tag*⟩. Important!

Codes For me two-part macros are at the heart, with one-part UI markup tags on top, one minimal. As can be seen below the two-part macros setup can be written without detailed knowledge in a functional spirit. Details are postponed.

```
\def\beginhead{\the\prehead
 \bgroup\headfont}
\def\endhead{\egroup\the\posthead}
%with auxiliaries
\prehead{\vskip0pt plus2ex
   \penalty-250\vskip0pt plus1ex
   \bigskip\noindent}
\posthead{\medskip\nobreak
   \noindent\ignorewhitespace}
```

The `\prehead` replacement text has been borrowed from Knuth, and `\ignorwhitespace` has been communicated by Phil Taylor at a BachoTeX some years ago.

8: This has all to do with how the OTR — OutpuT Routine — splits the main vertical list, MVL for short, and is a key issue of TeX's mapping copy onto pages.

The one-part markup form *with the same functionality* — nearly minimal because of the curly braces — reads

```
\def\head{\beginhead\bgroup
   \aftergroup\endhead \afterassignment
   \ignorespaces \let\dummy=}
```

Explanation. The replacement text starts `\beginhead` and a group. Immediately after the opening of the group `\aftergroup` takes care that the second part of the two-part macro will be invoked at the right place. The opening brace after the invocation of `\head` is read away by assigning it to `\dummy`. Spaces after the opening brace are ignored, due to `\afterassignment\ignorespaces`. The TeXnique — moving the group opening — has been borrowed from plain's `\footnote`, *The TeXbook* 363. There it is used at the end of the replacement text of `\fo@t`.

The straight minimal markup macro reads as follows.

```
\def\head#1\par
   {\beginhead#1\unskip\endhead}
```

Note that here the processing on-the-fly functionality has been lost at the expense of minimal markup.

Example 3. Use of head macro

In practice I use most of the time `\bluehead` with a blank line to separate the heading from what follows. But when processing on-the-fly is wanted[9] I use the more powerful lower level macros as shown below.

```
\beginhead|Head text
           in verbatim|\endhead
   Text after\\next line

\head{Head text, with |this|
      in verbatim}
   Text after\\next line
```

with results

`Head text in verbatim`
Text after
next line
Head text, with `this` **in verbatim**
Text after
next line

9: I personally don't like much the use of different fonts, especially not in the heading.

In general we have to store the title for reuse in a running head or a ToC, where the title is usually set in different fonts. When we store within TEX we have lost the processing on-the-fly capability. To store material in a file and read it again when needed maintains the processing on-the-fly functionality. Here we have touched upon another issue which should be known about (LA)TEX use. How are ToCs done? When no file is used one can't expect that material will be processed with suitable catcodes, that is with possibly different catcodes.

And the alternatives? They suffer generally from

- too many ways to provide arguments, and why should we allow arguments at all
- general parsing technique of arguments.

What to do as macrowriter when creating your own push-the-button level? Study these macro collections — it will take you some time for sure — or simplify matters and stay with Knuth and let the `\(blue)head` be followed in the markup by just the title and a blank line and that is it, for most if not all of the cases.

TEX inside in sight?

We have Knuth's — much overlooked manmac by the way — macros, AMS (and TUG) macros, next to LATEX, to name but three streams. These sets are highly incompatible. To use macros from AMS (TUG) is near to impossible because they are tied up with general approaches such as a common parsing philosophy, unless the macros which embody the basic philosophies are taken over as well, creating a too big an overhead not to mention the lack of understanding.

To adjust the OTR of LATEX is not trivial to say the least. I would not trust the result in general because I for example can't tell whether it has influences on something elsewhere.

The desease of (non-Knuth) macros was, and probably still is, that once a macro (set) has been released it is immediately followed by a newer release to adjust for the bugs or insufficiencies.

Powerful TEX features, akin to EP, have been overlooked at large. Let me mention only three items

- non-alien optional arguments functionality
- two-part macros
- active list separator.

Many a user got broke with the unusual `\expandafter`, `\futurelet`, `\afterassignment`, and `\aftergroup`. They are just coding auxiliaries to realize general functionalities, and of course a macrowriter should be aware of these, in short should know how to use the mentioned control sequences.

Optional arguments. This concept was known in high-level programming languages — HLPLs, for short — as well as in command languages, the pre-window one-liners to instruct computers. Maybe because of programmers being familiar with those, they have incorporated them and overlooked Knuth's new approaches. He has used token variables of which the text can be used — remember that TEX is all about text replacement — can be inspected without parsing overhead, as simple as that. Next to Knuth's `\every`... I introduced `\this`.... I used it on many places, for the first time in my `\btable` — bordered table — macros. There I abstracted into attributes such as `\framed`, `\(h/v)ruled`, and so on.[10]

EXAMPLE 4. NUMBERING LINES IN VERBATIM

How to implement the option of line numbering in verbatim? I did it by a definition `\numvrb` — called attribute in the SGML world, or simply flag — which can be set by its invoke in a `\thisverbatim`. Default `\nonum` is on. Next to this option we can think of other 'options', for example reading a file verbatim. And how to cope with semi-transparent verbatims?

```
\def\numvrb{\vrblin0
  \everypar{\advance\vrblin1
   \llap{\sevenrm\the\vrblin\quad}}}
%use
\thisverbatim{\numvrb}
\beginverbatim
...
!endverbatim
```

Note that BLUe's verbatim macros work on a line-by-line basis and that the line numbers can be

10: At that time I had not adopted a general philosophy about handling of options, the `\this`... approach. I adopted the latter when I developed BLUe. So, some non-uniformities are in BLUe. The few attributes in `\btable` seemed so natural to me, that I did not change those, at the expense of non-uniformity.

adjusted via modifying the counter \vrblin. The ! is the default escape character in BLUe.[11]

Knuth's handling of options A beautiful example of how Knuth handled options is in manmac's \begindisplay and \enddisplay. Not via \every...— Knuth's main way of coping with options — but differently. It is used in *The TEXbook* 29. All what follows after the opening tag \begindisplay up to the end-of-line is taken as '(optional) argument,' and inserted in the alignment display between $$ and \halign.

No curly braces nor other opening and closings like [and], or parsing. And when you don't need the option functionality you won't notice. It does not hinder or impose on you. Minimal markup? Yes, definitely!

```
\begindisplay\hbadness10000
\hbox spread-.666667em{The badness
   of this line is 100.}&
     \quad(very tight)\cr
...
\enddisplay
```

This example shows what Knuth had on his mind with respect to options and explains why he did not introduce \this⟨*tag*⟩s. The coding is a litlle more complicated but systematic.

Can it be applied throughout a format? With \beginverbatim I stumbled upon problems in applying options to the in-line verbatim with markup |...|. Needless to say that it went smooth via \thisverbatim.

Two-part macros. This concept was new for me until some 7 years ago. It does not occur in the HLPLs I'm familiar with. Knuth uses two-part macros but did hardly talk about them in *The TEXbook*. In LATEX's macros it is completely absent, at least in the LATEX I looked into some 10 years ago. In using two-part macros your personal implementation can become simpler. In pursuit of Knuth I have used them abundantly in BLUe. This is a consequence of the catcodes idea, especially how and when catcodes are fixed. It comes down to when you parse an argument the catcodes are fixed while reading the argument. With two-part macros catcodes are fixed on-the-fly. This comes in handy when creating verbatims with an escape character, the so-called semi-transparent verbatims.

In Chapter 20 of *The TEXbook* there is no treatment on two-part macros, nor is there an entry for it in the index, alas. Exercise 5.7 deals with named blocks and checking of them. The latter is used in LATEX to make sure that the right environment closing tag is used in the markup. In Appendix E, where example formats (o.a. manmac) are explained, two-part macros are abundant. From the Appendix the following.

```
\beginchapter...\endchapter
\beginlines...\endlines
\begindisplay...\enddisplay
\begintt...\endtt
\begimathdemo...\endmathdemo
\beginchart...\endchart
\beginsyntax...\endsyntax
\begindoublecolumns...\enddoublecolumns
\exercise...\answer...\par
```

Furthermore, in the past questions were posed on TEX-nl, which exposed unfamiliarity with two-part macros. All this was enough for me to spend a paradigm column on two-part macros.[12]

Example 5. \beginlines...\endlines

The functionality is that the script in between is processed line-by-line and the result is preceded and followed by an \hrule.

```
\def\beginlines{\par\begingroup\nobreak
   \medskip\parindent0pt\hrule\kern1pt
   \nobreak\obeylines\everypar{\strut}}
\def\endlines{\kern1pt\hrule\endgroup
   \medbreak\noindent}
```

In *The TEXbook* script this is combined with in-line verbatim.[13]

Explanation. The replacement text of \beginlines is processed, followed by the formatting on-the-fly of the inserted material (starting on the next line after \beginlines) up to \endlines. The replacement text of the latter finishes it up.

Unwanted breaks are avoided. The \hrule is set in the first part and in the second part. Opening and

11: The details of of BLUe's Verbatim have been explained in MAPS 94#1, available also on NTG's 4AllTEX CD-ROM.

12: See 'Paradigm: Two-part macros,' MAPS 95#1, available also on NTG's 4 AllTEX CD-ROM. Note that Knuth's \beginchapter does not process the title on-the-fly.

13: To set text verbatim. By the way, this is another approach to 'verbatims with an escape character.'

closing of the group must be done at the right place. The in between script is processed with `\obeylines` on.

Why? The need for bothering about two-part macros is that the enclosed script elements are processed on-the-fly, meaning with the right catcodes.

One-part vs. two-parts One-part macros are explained in Chapter 20 of *The TEXbook*. They are used as a shortcut for the replacement text parameterized by at most nine arguments.

A two-part macro is different. The first part sets up the 'environment' followed by script elements and ended by the second part, to finish up the environment.

LATEX emphasizes the environment concept — as a logical issue, with checking for the required endtag — in for example

```
\begin{abstract}...\end{abstract}
\begin{center}...\end{center}
\begin{itemize}...\end{itemize}
\begin{picture}...\end{picture}
\begin{quote}...\end{quote}
\begin{tabular}...\end{tabular}
\begin{thebibliography}...
          \end{thebibliography}
\begin{verbatim}...\end{verbatim}
%etc.
```

Example 6. Catcodes

To digress a little on the above the following hypothetical example about the use of `\begindemo*\enddemo` versus `\demo*`. The purpose is to demonstrate that * will show up differently. The former is implemented as a two-part macro with processing on-the-fly capabilities, and the latter is implemented as a straight (one-part) macro with an argument. Suppose we have

```
{\catcode`*=13
 \gdef\begindemo{\begingroup
   \catcode`*=13 \def*{MUL}}
 \gdef\demo#1{\catcode`*=13 \def*{MUL}#1}
}\let\enddemo\endgroup
```

then the result of

```
\begindemo*\enddemo
%and
\demo*
```

differs. The first yields MUL and the latter *.[14]

Explanation. In the two-part case all is straightforward. While in the one-part case the * as argument is processed with the catcode at the moment of invocation, usually differently from 13, a token different from the one which is redefined in the replacement text.

One-part on top Once we started from two-part macros we can build a one-part on top with the same functionality, albeit the tag must be followed by its argument(s) surrounded by curly braces. This can be achieved — also by TEX — in the following meta way.

```
\def\tag{\begintag\bgroup
   \aftergroup\endtag\let\dummy=}
```

Note that in the above example we could have complicated it more by introducing `\demo{*}`, with this `\demo` equivalent to the two-part macro.

`\eqalign` as two-part macro As an example of how to cast a one-part macro into two parts, and a one-part macro[15] on top, let us rewrite `\eqalign`, *The TEXbook* 362. The extra functionality of this approach is that the two-part variant can be used in those cases where the argument needs to be processed on-the-fly.

```
\def\begineqalign{\,\vcenter\bgroup
   \the\thiseqalign\openup1\jot\m@th
   \starteqalign}
\def\starteqalign{\ialign\bgroup
   \strut\hfil$\displaystyle{##}$&&
   $\displaystyle{{}##}$\hfil\crcr}
\def\endeqalign{\crcr\egroup\egroup}
%with the one-part
\def\eqalign#1{\begineqalign
   #1\endeqalign}
```

I don't have a concrete example for the need for modifying `\eqalign` towards processing on--the-fly.[16] However, it illustrates how to rewrite a one-part macro into two-parts as basis.

The active list separator. This functionality is used abundantly in BLUe, especially in selective loading of a tool, picture, reference, or address from a database. I used it for the first time in the Tower of Hanoi

14: A robustness aspect is to use `\begingroup` and `\endgroup` instead of `\bgroup` and `\egroup`, to facilitate the location of (user) unmatched braces in between.

15: Not more limited than the one available.

16: Although Knuth was not consequent, he apparently was right not to persuit things unduly.

problem to be played and set by TEX.[17] By the way Knuth used the active list separator in *The TEXbook* script, namely the `\ansno` to set the answers of the exercises after having written them first verbatim to a file.

Addresses, pictures and references The entries of these databases obey the syntax

`\lst\⟨name⟩{⟨entryproper⟩}`

that is the list consists of groups of two elements each preceded by the (active) separator.

Selective loading is achieved by properly defining `\lst`. Moreover, the names of the entries to be selected must be defined with whatever you wish as replacement text.[18]

```
\def\loadselectivefrom#1{%#1 = address
 \def\lst##1{\ifx##1\undefined\ea\gobble
         \else \ea\gdef\ea##1\fi}
 \input #1.dat \relax%e.g. lit.dat
 }
\def\gobble#1{}
```

Because of the scanning `\outer` defs are not allowed, nor are `\par`-s. The selective loading macro is embedded in the user macros `\references` and its ilks. In detail the meaning of 'loading' is adapted to the application. For references this means that the specified entries are set in a box and their names redefined by numbers. The names can be used for cross-referencing purposes while the box can be pasted up at your place of choice.[19] However, the underlying searching methodology is the same for addresses, references and pictures.

Example 7. Loading and using a picture

Loading a picture can be done simply as follows where I have abstracted from the filing system and therefore the markup is independent from the computer operating system.

`\bluepictures \⟨name⟩pic, ...`

The use is simply the invoke of `\⟨name⟩pic`, eventually preceded by `\thispicture{...}` to account for another scaling, the level of detail and so on.

How to create a picture for inclusion in the database ready for reuse is another story.[20] For reuse detailed knowledge is no longer necessary. All that has to be remembered is the name of the picture, how to include it, and how to adjust it, that is how to override the default settings.

Different font sizes. The LATEX world abstracted from the fonts via the use of the NFSS (New Font Selection Scheme) tool.

Again detailed knowledge is not necessary when one starts from Knuth's size-switching macro template, Plain TEXies should copy and edit Knuths size-switching macros — `\tenpoint`... — from *The TEXbook* 414.[21]

Only a few times I needed inside knowledge. There was one occasion where I really needed advanced inside knowledge. It was for typesetting partitioned matrices with braces along its sides of *different length* but *equal thickness.* Jörg Knappen drew my attention to this aspect and provided a real hackery solution due to Alan Jeffrey.

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \end{pmatrix}$$

On another occasion I was troubled how to hyphenate accented words automatically and *completely*, especially the part after the accent is not hyphenated with the default accents. Bernd Raichle and Bas Romeny provided satisfactory explanations and ways around. The explanation is that an accent introduces *non*-implicit kerns before and after, and hyphenation stops at such kerns and therefore the rest is not hyphenated.

17: For more details see 'Paradigms: Searching,' MAPS 96#2, available also on NTG's 4AllTEX CD-ROM.

18: This approach is the opposite of preventing reloading. We tacitly want to redefine the fancy entries by the meaningful ones. My fancy text to be replaced is an error message.

19: Provided the box is not too large, otherwise it should be unboxed first.

20: See the PWT guide, or the note Paradigms: The winds and halfwinds, MAPS 96#1, also available on NTG's 4AllTEX CD-ROM.

21: It is said that scaling does not yield the best results. For optimal results we should run METAFONT again with adjusted parameters. TEX's inside knowledge should be complemented with knowledge about METAFONT's inside — and some extras to find out and decide about what is optimal.

But, one can with inside knowledge appropriately redefine the accent in such a way that TEX3.... will hyphenate the complete word.

Conclusions

I think for the easy use of (LA)TEX with TEX toolboxes and macros knowledge of TEX's inside should not be necessary. Insight in TEX is mandatory, alas. Too much of this is the Achilles' heel for real wide-spread use of (LA)TEX.[22]

Insight issues which have to be known are

- tokens are characterized by ASCII-catcode pair
- I/O abstraction
- the boxes, glue and penalties approach
- linebreaking works over paragraphs (how paragraph parameters work)
- how copy is reused (stored within TEX or within a file)
- awareness of (rigid) discrete sized fonts sets.

For macro writers the following extras

- processing on-the-fly via two-part macros
- the concept of the active list separator
- the use of token variables for options.

Because of the above an easy, push-the-button, robust and reliable use of (LA)TEX and its accompanying tools is not yet in sight. Even worse, most likely I'm incomplete about what has to be known.

The add-ons don't compensate for this either, because they comprise generally mammoth collections, with the absense of small, concise and confined macros which perform one well-defined task to be combined into bigger tasks.[23]

History has it that the latter has been recognized as useful in creating numerical program libraries, in UNIX' little languages and pipes, and in the RISC machines.

To make the circle round knowledge of TEX's inside is mandatory for those who go for designing and creating the successor of TEX, the ϵ-TEX casu quo NTS[24] activities.

Because of the inherent complexity of the typographical tasks, a push-the-button successor of TEX is not yet in sight.[25]

Make your choice: *either master TEX inside, or gain TEX insight.*[26] My case rests. Have fun, and all the best.

◇ Kees van der Laan
cgl@rc.service.rug.nl

22: Of course, we all take this for granted because of the lack of a comparable or better alternative.

23: I know, people would object and say that there is somewhere a LATEX style file for so and so. But that is not general and restricted to LATEX. I don't like LATEX's design, nor do I wish to use it. I have created my own set which I understand, which is invariant to a high degree and suited for the purpose of the markup and typesetting of my notes for the rest of my life. TEX is my EP lingua franca, comparable to English as a communication tool.

24: NTS stands for New Typesetting System initiated by DANTE. e-TEX — evolving TEX — is the more modest and realistic spin-off approach towards the upward compatible successor(s) of TEX. Both projects are directed by Phil Taylor. Peter Breitenlohner and Bernd Raichle are at the heart of e-TEX, to name but two knowledgeable wizards on the project. This makes me believe that it is a worthwhile effort, that a useful result will see the light of day some time. However, I can live with TEX-as-is and the proposed changes I have heard off so far are not really necessary for me. It is a well-known social law that when an ϵ improvement is done it does not provide enough reason to follow the transition because of costs entailed by change, if not for the incompatibility disadvantages. But as a research/development project it is mandatory. For a status report see Phil Taylor's report for example in *TUGboat*, 18.1.

25: For me a TEX script as a *useful* hypertext is not yet in sight either. Of course, the idea of a hypertext is nice, but I prefer, for the moment and near future, the selection and composition done by an author, instead of jumping through hyperspace and get some meaning out of it. However, jumping from a ToC or index entry towards the appropriate place is nice for a reader. As author I use ToCs differently. A ToC provides me a means to watch over structure when writing on-the-fly, as I usual do, starting from vague ideas of what I have on my mind. By writing them down and elaborating on them they become explicit and ready for development, communication and discussion.

26: *At BachoTEX'97 I learned another wordplay from Phil Taylor: 'incite,' for example 'insults incite resentments,' or more to the point 'feedback incites improvements.'*