Zpravodaj Československého sdružení uživatelů TeXu

Hans Hagen

Some NTS thoughts

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 9 (1999), No. 3, 109–115

Persistent URL: http://dml.cz/dmlcz/149841

Some $\mathcal{NTS}$ thoughts

Hans Hagen

The next stage

When we take a look at Peter Breitenlohner's ε-TEX, we see extensions in the spirit of TEX. Based on experiences with macro writing, some limitations are removed (more registers like `\dimen`), some optimizations have taken place (like `\aftergroup`), protection is introduced (`\protected`), there are some more conditional, expansion and scanning features, and when we look at the typographic engine, bidirectional typesetting has been added. In extending TEX a clear distinction has been made between what should go into ε-TEX and what into $\mathcal{NTS}$.

When we look at Hàn Thế Thành's PDFTEX, we see a new backend. This in itself is rather revolutionary, because there are strong movements to stick to DVI and use this format as the intermediate to others. It is also revolutionary, because it removes the postprocessing stage, and thereby forces macro packages to take care of everything themselves, instead of relying on the post-processor. Last of all, we see that PDFTEX introduces an additional paragraph break pass, using infinitesimal horizontal glyph scaling.

Taco Hoekwater has written an extension to ε-TEX that introduces extensive list manipulations that can be of help when writing parameter driven macro packages, as well as provides an alternative input parser, targeted at SGML. His extension does not concern typography.

When we summarize these developments, we can conclude that ε-TEX has smoothed the path to $\mathcal{NTS}$ by providing a suitable arena for discussions. At the same time PDFTEX opened the road to a more drastic deviation from traditional TEX, if only by boosting TEX into the next century where graphics and interaction will dominate. Taco's TEX makes clear that we cannot neglect the input side and should closely follow the developments in the SGML field. The lesson learned is that we should not be too afraid to extend TEX.

In practice only a few people extend TEX in the ways mentioned, and most users will fall back on macro packages that hide most of the details. This aspect cannot be neglected when we look into the future of $\mathcal{NTS}$. Extensions like those provided by PDFTEX can rather well be supported by macro packages, because they often have a concept in which different drivers can be used. The level to which the specific features are supported depend however on the flexibility in the lower level shell. In TEX object reuse is limited to boxes and not reflected at the output level, opposite to PDFTEX. But downward compatibility can be guaranteed to a great extent because in most cases duplicates can be used instead. Moving from TEX to PDFTEX is therefore rather smooth.

When we want to use some ε-TEX features and at the same time want downward compatibility, we are forced to provide (often) poor man's alternatives or fall-backs. For extensions like ε-TEX to be accepted, it's best when all users of a macro package switch to this extension at once. With extended ε-TEX, downward compatibility can be achieved by providing preprocessors or clever but slow macros. So here, massive adaption is less important: use it when you need it.

In this respect the lesson learned is that a change should be guided and guarded, especially when many users depend on the macros other people provide. Before I will go into more detail into some challenging extensions that $\mathcal{NTS}$ can bring, I want to set the framework in which such extensions can take place.

If we value $\mathcal{NTS}$ as just a re-implementation, the project is nearly finished. In about a year the monolithic PASCAL code is transformed into highly structured JAVA, well tested and documented. But right from the start the project team was more ambitious. Of course, when the first version is available, everyone can take the source code and start writing his or her own Yet Another New Typesetting System, but the large body of the current TEX users will quite certainly benefit from some coordination. Ideally the situation around TUG 2000 conference will be as follows:

- There is a robust $\mathcal{NTS}$ environment, that users can easily download and plug into their JAVA enhanced operating system.
- The $\mathcal{NTS}$ project team has set up an infrastructure to host discussions, if possible centered around topics, identified in earlier stages of for instance the development of ε-TEX.
- To enable a smooth adaption of $\mathcal{NTS}$ without losing existing functionality, existing extensions to TEX will become available as plug ins into $\mathcal{NTS}$.
- Extensions are discussed, prototyped, validated, developed and documented in a professional context conforming standards laid down by the project team.
- The $\mathcal{NTS}$ project team will guard consistency and take care of proper hosting of new functionality.
- Macro packages will adapt new features in such a way that continuity is guaranteed for the large TEX user base.

Of course, use and reuse of code by whoever wants to do so will be stimulated. But, one of the strong points of good old TEX is that it is stable. The challenge will be to provide stability as well as flexibility. Imagine a macro package providing five different multi-column methods, each demanding different low level features. Ideally your system will pick up the right (and latest) code needed to use the particular method.

Think of this ...

When the first official release of $\mathcal{NTS}$ becomes available, we can start thinking about implementing all those features missing from TEX. How about a more procedural programming language, name spaces, string manipulations, more advanced error recovery and loading patterns on demand. Sure, these are all worth thinking of, but the typographic extensions will be the most challenging. Here I will discuss some ideas (some of which will be demonstrated in the presentation). Will you stick to using TEX or will you move on to $\mathcal{NTS}$? Maybe the next paragraphs will convince you at least to closely follow what is happening.

Columns and grids

In spite of all efforts, nobody has come up with the perfect multi-column output routines. This is mainly due to the lack of proper support in the kernel of TEX the program. Before computer typesetting came around, much manual effort went into typesetting documents in columns, and translating more intuitive behaviour into an algorithm is not trivial. A few decades of TEX at least have taught us where the complications lay.

- First of all we want our columns to be perfectly balanced. This is trivial for pure text, but imagine lots of white space, like display math.
- We want floats to be moved to the best available location. Of course we want floats to span more than one column, and even spanning one and a half columns with a text flowing around the figure should be possible.
- In double sided output, we want lines to align on the opposing pages (spread). When we hold the paper towards a bright source of light, we want the lines on both sides of the paper to align too.
- We definitely don't want to end up with a few lines or words on the last page. Why not apply a small percentage of glyph scaling in such a way that we get full pages? Of course we will need more than paragraph and page optimization for this: we are dealing with the document as a whole.
- Columns may differ in width. Think of two columns, spanning one third and two thirds of a page. In the middle of such two columns we will want to typeset an illustration, and the text should follow the circular shape of this illustration.
- Talking of illustrations, instead of being something with fixed dimensions, the scale may be adapted, of course consistently, to suit the overall document appearance (grid, spread, and more).
- Are you still thinking from left to right? Text can go in all directions, and will be mixed too. The width of columns may change in the meantime. Anyone who has seen traditional Jewish religion documents, will see the challenge in nested columns with (foot)notes flowing around partial columns.

Do you know a macro package that offers this functionality? Some day there will be packages around, build on top of basic $\mathcal{NTS}$ functionality. The main question is, what and how should $\mathcal{NTS}$ provide this functionality. We don't want Just Another Desktop Publisher, because those are around already. We want the building blocks and want to pack them into meaningful, clever, semi--intelligent macros. Will we still talk of macros? Some of the things mentioned are not too difficult to program, but making the combination behave well is the challenge.

The overall appearance

TEX is famous for its breaking of paragraphs. Actually it is still famous for that, simply because the main stream word processors think in lines. In spite of the—in the eyes of some users—perfect line breaks, users and implementers have been in search for even better ways of dealing with paragraphs.

It is uniform greyness that we want. The more uniform it is, the more comfortable it reads. TEX's paragraph builder can be influenced upto a certain extent, but even experienced users cannot foresee how a combination of settings will influence the look and feel. So what will we need?

- Typesetting is more than manipulating metrics. Don't we need a typesetting system that looks at the glyphs themselves, the small graphics?
- People tend to disagree on what looks best, but experts often agree on what looks worse. Why not build in expert knowledge, or even better, build a system that learns from the user's rating?
- How is greyness calculated? Does $\mathcal{NTS}$ act upon the internal lists of glyphs, or does it first build a bitmap? At least then it knows how the pages come out. Is the validation a function of an output device? Will the shape of glyphs depend on the rating? Will TEX and METAFONT become one?
- Is, in validating the appearance, a model of the page needed, in terms of meaningful areas? If so, how is such a model defined? Do we need pattern recognition?

There is some experience with manipulating glyphs, using either a large set of alternatives or manipulating (stretching or shrinking) glyphs. But far more experimenting is needed in this area. What better tool to use than $\mathcal{NTS}$, where we can plug in a new mechanism without spoiling the rest of the system? Rivers, dirty skylines, big emergency stretched holes, they will all disappear with the years to come. Unless of course the migration from paper to screen has made us loose the feeling for details. And wasn't it the details that made Knuth develop TEX?

Embedded graphics

Ah, who does not want graphics these days. One of the strong points of TEX has always been the separation of isolated graphics from the document source. But at the same time, TEX's repertoire of graphic primitives is limited to those needed for building math glyphs: there are only horizontal and vertical rules. We want more.

METAPOST has demonstrated that combining TEX with graphics can be very stimulating. But at the same time, it demonstrates that exchanging information between a typesetting engine is far from trivial, especially because we are dealing with different concepts (and states).

- $\mathcal{NTS}$ needs a graphics engine, or maybe even several. Models for exchange of information between processes dealing with pure typesetting and drawing shapes need to be developed. Such mechanisms should cooperate naturally with the paragraph and page breaking as well.
- Typesetting along curves, turning shapes into outlines, and applying arbitrary filling and shading, it all makes sense.
- TEX is strong in math, but how about (bio)chemistry? Although satisfactorily results can be reached, more is needed. Haven't we all seen documents that made us wonder how to typeset that in TEX? Lots of thinking needs to go into that area.
- For some languages pasting together glyphs is not enough. Actually drawing glyphs, or even better: words or sentences can be an alternative. Even emotions can make it into typeset text. Strong handwriting oriented graphics has to meet expressive coding.

Layers in text

It was already showing up in DVI viewers quite early: searching through typeset text. As long as we are dealing with non-composed characters, searching is not so much a problem, but TEX is used for more than typesetting English. The internet has stimulated world wide searching and general mark up languages are used to enhance this. Formats used for dissemination of typeset text, like PDF, are already a bit more prepared for searching. How will developments like this influence future typesetting?

- First of all, the new system needs some more understanding about the typeset text. Support for UNICODE, unified glyph names is mandate.
- When searching through a document, some knowledge on what in language the text we're dealing with makes sense. Not only the (many) language(s) of a text, but the direction also plays a role. Complicated ligatures should be recognized properly.
- In more dynamic documents, like fill-in-forms, interaction with a typesetting engine is not a luxury, especially not in European and Eastern

languages. $\mathcal{NTS}$ can be such a plug in, but the document itself should contain the information needed to let $\mathcal{NTS}$ to do its task. A document is more than a collection of graphics and glyphs, and typesetting more than organizing those.

- As PDFTEX already demonstrates, using TEX to embed typeset information like pop-up-help and tool tips is a breeze. Although heavily dependent of features of viewers, $\mathcal{NTS}$ will benefit from a decent model of layers on which we typeset as well as concepts of information hidden in the output but showing up at wish.

Just in case one wonders what gadgets like tool tips have to do with typesetting, think of a tool tip that has to pop up left or right aligned, depending of the position on the page. It does not make much sense to let 10 centimeters of tool tip disappear into the margin.

Conclusion

I did not mention things like colour. Implementing colour is rather trivial although different models are possible. Will we think in colour as an attribute to a glyph? Or a word, or sentence? Is there something like a background, and if so, what is the background of a line? As long as models don't conflict, they can co-exist peacefully. This is unlike different views on paragraph breaking can conflict and lead to unwanted compromised or dead-locks. It will be clear that implementations, $\mathcal{NTS}$ the program, macro packages, and distributions, should all cooperate smoothly to make $\mathcal{NTS}$ a reality.

From ε-TEX we have learned that even when ideas are circulating in our mind, making them explicit and ready for implementation is not trivial. Discussion, conception and implementation takes time. It is no problem to come up with some quick and dirty hacks, but we want to go for the best. It will not be a one year job. ε-TEX has demonstrated that a well documented and structured PASCAL program can be extended to great length.

From PDFTEX we have learned that extensive experimenting has its value. Adding features, but also removing them when not useful or not fitting into TEX's way of dealing with things, has its benefits. The nice thing about PDFTEX is that there are many users who are involved in the testing. Because most changes only affect the low level interfaces of the the main stream macro packages, common users are not that aware of the many new primitives that are needed because specials have become obsolete. PDFTEX has also demonstrated that extending TEX in its current incarnation has reached its limits. It's an interesting breed of PASCAL and C, that incidentally proved that incorporating ε-TEX functionality was more easy than expected.

From extended ε-TEX we can learn that developments can be driven from need. Direct interpretation of SGML was needed and therefore written. Minimal

discussion sometimes pays off. Being an extension not related to output (DVI, POSTSCRIPT, PDF) or specific typesetting, it proves that extensions like that can be hooked into a typesetting system without disturbing and breaking existing code.

One may wonder if $\mathcal{NTS}$ will make *those other TEX's* obsolete. Given that it takes time to come up with real new things, we can be sure that the predecessors of $\mathcal{NTS}$ will be around for a long time, if only because they have virtually no bugs, are supported by macro packages and most of all do their job well. This gives the $\mathcal{NTS}$ developers the time needed to come up with real good concepts and implementations.

Given that the TEX community has demonstrated that extending a stable program is feasible without harming existing, often critical, typographic production processes, $\mathcal{NTS}$ has a great future, although, in many areas, we haven't yet reached the limits of traditional TEX. It takes time and discussion, but talking of life-long tools, we have some time left. It is up to the $\mathcal{NTS}$ team to stimulate and guard this process, and we hope we will not fail you.

It sounds like I'm believing it myself, doesn't it?

Hans Hagen
PRAGMA Advanced Document Engineering
Ridderstraat 27
8061GH Hasselt
The Netherlands
E-Mail: pragma@wxs.nl

$\mathcal{NTS}$: nový sázecí systém

KAREL SKOUPÝ

Počátky projektu

Diskuse o potřebě dalšího vývoje TEXu či vytvoření jeho nástupce se rozvinula již počátkem devadesátých let. Vedly k tomu důvody technické i politické. Politické důvody by se daly velice stručně vyjádřit jako potřeba udržení a zvýšení atraktivity TEXu pro zachování a rozvoj komunity uživatelů. Více informací o tom lze najít zejména v [8].

Technické důvody spočívají především v požadavcích na ještě vyšší kvalitu a rozsáhlejší možnosti zpracování dokumentu. Týká se to například sazby ve