Bijlage 2

TEX in 2003: Part I

Propositions and conjectures on the future of TEX

NTG TEX future working group
P.O. Box 394,
1740 AJ Schagen,
The Netherlands
ntg-toekomsttex@ntg.nl
http://www.ntg.nl

Introduction

In the last year, there has been a lively discussion within the Dutch TEX Users Group about the future of TEX. This discussion was initialized by a couple of posts to the TEX-NL e-mail list by Hans Hagen and Taco Hoekwater, but it soon spread to a much larger group of correspondents.

Eventually, this resulted in a meeting between the most interested people in December 1997. The current articles are a re-working of the long-term proposals and requests formulated by this group of people. The short-term requests were passed on to the eTEX team.

Our views on current work

At the moment, there are at least three distinct projects available to current TEX users that are working to extend TEX: Omega, Pdftex and eTEX.

The first two of these are in a sense niche products: If you don't need either non-latin language typesetting or PDF output, there is little point in learning how to use these two programs. The third project, eTEX follows the more general approach, and is potentially of interest to every current user of TEX.

The work done in eTEX is nicely thought out, and the result is both stable and virtually bug-free, but it is hardly ever used in real applications. The reason is simple: package writers will not use eTEX primitives until they can be certain that eTEX is indeed available everywhere. On the other side, eTEX cannot develop without input from package writers that intent to use eTEX. There is a chicken–egg situation, and it leads to the following conclusion:

1. *eTEX is a nice idea with too little momentum to make a difference.*

Another important problem is the fact that people that need the functionality of either eTEX *or* Pdftex *or* Omega *and* one of the other two extensions, cannot do so from within one document. All three have their own specific syntax extensions, that are hard to fake in one of the other extensions. This is unsolvable in the current situation, and leads us to the following statement:

2. *Omega, Pdftex and eTEX should be merged as soon as possible.*

Then there is a fourth project that has just started: the New Typesetting System (NTS).
The NTS development group hopes to increase the chances of general acceptance of NTS by guaranteeing compatibility with TEX for a number of years to come. We feel that this is a error, because most of the more fundamental issues that NTS should deal with to live up to the 'New' in it's name cannot be done without sacrificing that compatibility. Issues like grid-based typesetting and better insertion control are very likely to require a completely new algorithm, resulting in a completely new implementation. Of course it is possible to do these things in parallel, but trying to implement something new while having to be really careful not to break the old implementation unnecessarily complicates development: people that want to use TEX should stay with TEX anyway.

Therefore, we urge the NTS group to reconsider their decision to stay compatible with TEX for at least the next five years.

3. NTS *will be pointless if it intends to be compatible with TEX82*

The second remark we have deals with the proposed modularity of the system, which is facilitated by the use of Java:

4. NTS *is a step forward and a step backward at the same time.*

A great feature of NTS will be its extensibility. This is similar in many ways to current LATEX packages, albeit much more advanced. Since NTS will be written in Java, one can

easily extend NTS with it's own classes. We presume there will be an easy interface to extend NTS (if not, someone will just hack the sources).

In all likelihood, this will result in precisely the same problems that current LaTeX has:

- Users are not aware of the packages available, and so keep asking questions like: How can I make this work in LaTeX?
- Furthermore, the portability of source documents (the .tex or .nts? input file) will be seriously endangered. We expect to see things like:

```
Error: this .nts style-file requires
module x.y which has not been
installed on your system.
```

The NTS team should give very strict rules for these extensions, otherwise we'll end up with another `\special`- similar situation. A central registry and a "head maintainer" are needed to keep track of extension modules in order to prevent these problems. It would be wise to turn this work into a full-fledged job under the control of (probably) TUG.

5. *We need time to experiment and must not fall into the every year a new version trap.*

An interesting common aspect in all current work is that only experience can lead to useful functional specifications. It is likely that NTS functionality will follow the same track.

This means that when we deal with the next generation TeX programs, common users must be patient until the developers of extensions and macro packages trust the new features and can guarantee upward compatibility. It also means that it will take some years until eTeX as well as NTS will be accepted as descendants.

We have to keep in mind Knuth completely rewrote his first TeX!

Packaging of Distributions

Over the last 5 years TeX has become a lot easier to install. The most important reasons for this are:

- Cd-roms have become available at large. These can easily hold a complete TeX system. The old-fashioned piles of diskettes gave far too much trouble, and tape is only for professionals.
- Recently hard disk space has become so cheap that complete installations on hard disk are not unusual anymore.
- Installation scripts were made to shield users from tedious setup and configuration issues.

Still a number of problems remain because they are inherent to the way that TeX systems work:

- A typical TeX system consists of an incredible number of files (more than 31415). No one really knows which parts are essential and which parts are not. In other words: every system is too large.
- "Everything" can be found on CTAN but only the most recent version. Old versions can be necessary to run old documents. Old CTAN dumps on cd-rom can be used to track down older versions, but we really need more professional version control.
- Maintenance is only feasible for professionals. Others are better off replacing the entire system, even though this will undoubtedly cause problems. The draw-back of 'plug & play' systems is that users have no idea anymore of the inner workings of the system. Is that a good thing or a bad thing?
- There is no such thing as an easy upgrade path. It's usually very hard if not impossible to simply add some files to a system and make them cooperate.
- Initial configuration can be automated, but reconfiguring is usually very hard. Any typical TeX system contains dozens of configuration files in almost as many completely different flavours. As a rule they are scattered all over, and only an absolute expert can deal with this.

This leads to a number of conjectures:

6. *The number of files in a typical TeX system should be reduced by a factor 100.*

We can achieve this by redefining the way any program finds its resources. A central database should be queried for any resource. This database should physically contain all resources. And of course it should be able to report (in any required level of detail) what's available. The database may event connect to CTAN (another database application) to retrieve resources not available locally.

This setup would allow for a minimal local installation to grow as necessary using Internet.

7. *Configuration of a TeX system should be centralized and automated.*

If we can realize the previous issue this one will not be too hard. Programs should specify formal descriptions of the configuration details they need. These could then be generated through menus or automatically by scanning the current setup, i.e., querying the database.

8. *Installation and maintenance should require far less expertise.*

The database may occasionally query CTAN for any updates. The administrator would get short descriptions of these, with links to complete documentation. He/she could then select which ones should be installed. This could even be done silently (overnight) if you want an up-to-date system all the time. If necessary, programs will be signaled to reconfigure themselves.

This setup should also take care of the endless problems with non-portable DVI files. We should all be using the same resources and if we are not, the system should warn us about possible mismatches. If we decide to make TEX produce DVI files that require no virtual fonts at all (i.e., TEX reads VF's itself instead of the DVI driver) an important source of problems can be eliminated.

9. CTAN *should have a complete index with descriptions of everything and cross-links to anything related to anything.*

This is obvious now if we want the systems to interact. Uploads to CTAN will have to be checked more carefully: descriptions, specifications, version number, relations to other packages, dependencies on other resources, etc. must be supplied. Any item that doesn't comply to this convention should be moved out ('not supported') and deleted after a certain period.

We realize that this might cause a cultural shock in the TEX world, but we feel this is necessary to keep TEX alive & kicking in the next millenium:

10. *Anarchy is what made TEX great, and it's anarchy again that will kill TEX.*

Let's try to prevent this!

On-line Publication Wishlist

With the increasing growth of the internet, a whole new branch of documents has appeared: documents that are only or primarily intended for screen viewing. The used formats differ, but it is easy to see that there are some common issues involved in all of those: file download

sizes, hyperlink support and ease-of-use are important points for all of these formats.

11. *TEX is rather well suited to cater for those needs as it is, but some extensions are needed to make sure that TEX will stay/become in the leading position in this arena*

For about 15 years TEX was only capable of producing DVI output. The limitations in both TEX and the DVI format mainly concerned direct graphic support and color typesetting, but color printers were rare and the lack of graphics support could be worked around.

Although originally TEX was more or less supposed to handle everything itself, those 15 years of use have demonstrated that many applications, like color and graphic inserts, heavily depend on the DVI postprocessing stage. To a large extent, this is not feasible nor desired in on-line publication. On-line formats are all rather device independent themselves: otherwise people would have to publish several versions of the same document.

Theoretically, both pdftex the current trajectory using and DVI to PDF processing through dvips and the Distiller can offer similar functionality, given that post processors are available to help out in the second case, but we can imagine both methods drifting apart, and we feel that the use of external programs to solve intrinsic problems adds a great deal of unnecessary complexity to the system.

12. *On-line publishing needs primitive support*

In fact, most of the conceptual extensions like hyper referencing can be implemented using DVI and `\specials`. However, usage can be far more robust in e.g. current pdftex, simply because hyper referencing is build in, and there is no longer a need to run various programs in turn. The same goes for object reuse, fill-in forms, scripting (Java), and graphic inclusion.

But systems like pdftex also create new problems. Take for instance graphics inclusions: where originally TEX macros only had to bother with the dimensions of the needed box, on-line publishing backends have to include the file directly.

Another conceptual extension is hyper referencing. Although clever tricks can give acceptable results, all approaches based on current TEX interfere with either the explicit wishes of the author or the line and paragraph break mechanisms present in TEX.

13. *TEX objects should be easily re-usable*

When we look at object reuse, we see that this concept never surfaced in DVI (using `\specials`). This is probably due to the fact that especially screen designed documents need these features, and it hardly matters for paper output.

¿From the users point of view, reuse may look rather straightforward (a sort of variant on copying boxes), but from the implementors eyes, object definitions are just another interfering kind of <*whatsit*>. And why is it interfering? Simply because TEX has no particular mode which

suppresses all interference. Yes, we can use a box, and we can let things happen at certain locations in the document that don't do any harm, but the situation is far from optimal.

When applied to for instance figure inclusion, reuse can quite easily be implemented in original TEX (pure DVI, using Gilbert's DVIview), the traditional DVI–dvips–Acrobat trajectory or Thanh's pdftex. But PDF fill-in fields support demands for more.

To give you a real life example where objects are needed: in PDF one can define a check field with several appearances like on, off, mouse down, etc. Technically this means something like this (in pdftex syntax):

```
\setbox0=\hbox{$\star  $} \pdfform0
        \edef\on  {\the\pdflastform}
\setbox0=\hbox{$\bullet$} \pdfform0
        \edef\off {\the\pdflastform}
\setbox0=\hbox{$\times $} \pdfform0
        \edef\down{\the\pdflastform}
```

When defining the check field, we then can refer to `\on`, `\off` and `\down`, as in the following code:

```
\pdfannot{ ... /On \on\space0 R ...}
```

Currently pdftex only flushes forms to the output file when are accessed. (this feature is needed because we want to be able to try out things, without ending up with redundant objects, like in a macro that tries three different methods and takes the best result).

Back to the three objects, these won't end up in the file when we refer to them in the field definition above, because the field definition is handled like a `\special`: pdftex just passes the information through.

Therefore, we end up with invalid references: the object is referred to, but never passed to the file. What do we learn from this:

14. *TEX needs a real object model.*

One with immediate as well as deferred definitions, that do not interfere with the internal lists that TEX builds and that permits forward *and* backward referencing.

Another typicality that surfaces often in on-line documents is the fact that screen layouts tend to use a lot more page decorations and colors than traditional typesetting. This is an area where a lot of disagreement is possible, but in the real world there are lots of practical applications of this.

At TUG97 there were several presentations on graphics. The related discussions invoked a BOF session on graphic primitives. Direct inclusion of METAPOST output (in pdftex) had already proven that a relatively small subset of PostScript primitives could be used for advanced graphics and therefore the discussion focussed on those primitives.

These graphic primitives in TEX are not meant for drawing free hand graphics like one would do in programs like Illustrator, CorelDRAW, or indeed Freehand. Instead, they are most often (to be) used for things like visualizing statistical results, plotting functions and drawing almost-mathematical shapes that can be used to emphasize certain layouts. In these graphics, text plays a important role, and this text must preferably be typeset by TEX. It follows that inclusion of an external file will not do, and the conclusion is:

15. *TEX needs a reliable system for in-line graphics and colors*

The most important outcome of the '97 BOF session was an agreement on the way to go: define a set of extensions that permit direct METAPOST output inclusion. It was felt that this set could also suffice the needs of the mainstream graphic macro packages written in TEX.

During the NTG 'future of TEX meeting' the participants made the exact specification of these graphic primitives (currently to be implemented as `\specials`) one of its main goals. To this end, we had to create a formal specification of the syntax involved, and that put us right in the middle of the `\special` problems.

Our final proposal on that matter will appear somewhere else in these proceedings, but Gilbert has already done some of the groundwork. Below is his explainatory text on the `\specials` that are currently included in DVIview. This text is kept here because it demostrates very well that only a few primitive commands are enough to give almost full in-line graphics capabilities.

To allow for instance METAPOST drawings te be inlined in TEX you need several things:

- A macro to interpret METAPOST's POSTSCRIPT output. Hans Hagen wrote a set of macros for PDFTEXusing `\pdfliteral` commands. These macros are easy to adapt to another standard using `\special` syntax
- A primitive sub-set of POSTSCRIPT commands is needed. METAPOST uses only a few POSTSCRIPT commands to draw it's figures.

To actually test the inline graphics standard we needed a viewer where this support was easy to include. DVIview was coming to life at that time so it was logical to use that as a test and development environment.

All primitives are easy to interpret, except for a few things like clipping and the like. The syntax will probably change in the future when the new special syntax is standar-

ized. Converting these specials to POSTSCRIPT output (e.g. modifying dvips) is easy to do, since the commands hardly need any translation.
Specials and stuff for inline graphics in DVIview:

```
\special{dv:startgraphic}
\special{dv:stopgraphic}
\special{dv:moveto x y}
\special{dv:lineto x y}
\special{dv:curveto x1 y1 x2 y2 x3 y3}
\special{dv:stroke}
\special{dv:setlinejoin j}
\special{dv:setlinecap c}
\special{dv:setdash offset values}
\special{dv:setlinewidth w}
\special{dv:setmiterlimit m}
\special{dv:rotate r}
\special{dv:translate x y}
\special{dv:concat x1 y1 x2 y2 x3 y3}
\special{dv:newpath}
\special{dv:closepath}
\special{dv:clip}
\special{dv:fill}

\special{dv:gsave}
\special{dv:grestore}
```

As you can see the amount of commands needed to support METAPOST output is in fact quiet small.
Some explanations:

dv:startgraphic
Starts a graphics figure. It saves the current position and context of the DVI interpreter. The current location is marked as $(0, 0)$. As in POSTSCRIPT positive x, y draws to the right and up.

dv:stopgraphic
Stops a graphics figure and restores the context.

dv:moveto x y
Moves the current position to x, y.

dv:lineto x y
Draws a line to x, y. This does not actually draw the line but only remembers the coordinates. The actual drawing is performed by `stroke`.

dv:curveto x1 y1 x2 y2 x3 y3
Draws a Bézier curve starting at the current point to $(x3, y3)$. The control points are given as $(x1, y1)$ and $(x2, y2)$.

dv:stroke
Performs the actual drawing using the current pen-style, color and width.

dv:setlinejoin j
How lines are joined. `j` can be 0, 1 or 2.

dv:setlinecap c
How the line-endings will look like. c can be 0 1 or 2.

dv:setdash offset vals
Sets the pen-style. `vals` is any number of values and specifies how long the pen is on and how long the pen is off. `offset` can be used to specifiy a starting offset in the `vals` pattern.

dv:setlinewidth w
Sets the thickness of the current pen.

dv:setmiterlimit m
Sets the miterlimit.

dv:rotate r
Modifies the current transformation matrix so that everything following this is rotated `r` degrees.

dv:translate x y
Modifies the current transformation matrix so everything following this is translated (x, y).

dv:concat x1 y1 x2 y2 x3 y3
Multiplies the current transformation matrix with the given values.

dv:newpath
Discards any present paths and start a new path.

dv:closepath
Closes the current path. After this you can use fill to fill the closed path.

dv:clip
Selects the current path as the clipping path. All subsequent fills and strokes are clipped to the this path. The clipping path may contain one or more closed paths.

dv:fill
Fills the current path with the current color.

dv:gsave
Saves the graphics state.

dv:grestore
Restores the graphics state.

dv:setrgbcolor r g b
Sets the current color. `r`, `g`, and `b` are specified from 0 to 1.

dv:setcmykcolor c m y k
Sets the current color.

dv:setgray g
Sets the current gray-level. 0 means black, and 1 means white.

Though it is easy to extend this set and include much more POSTSCRIPT operators, this is not the intention. It should be noted that complex graphics which require the full POSTSCRIPT set of commands should be done by including the EPS file and let PostScript do the work.

Language extension wishlist

Removal of limitations regarding fonts

The font limitations that are inherent in the TFM format should be dropped. One fairly simple way to achieve this is to make TEX read `.pl` or `.vpl` files instead of TFMs, but it is also possible to adopt a new format like Omega's OFM files or even create a completely new specification.

An overview of limitations in current TEX shows limits in almost all places: the amount of characters present in a TFM, The number of separate width / height / depth / italics-corr values, the number of ligatures and kerning pairs, math sizing stuff, etc. Almost all of these limitations are not really needed anymore; most of them were born out of Knuth's desire to use as small an amount of memory as possible.

Especially the current implementation of math mode has some really weird demands on the used fonts (some characters get really weird places in the glyph container, like integrals and delimiters are all below the baseline, and the height of the `\sqrt` sign is used to decide the width of the extension bar). This should be fixed so that it becomes possible to use non-metafont math fonts in a reliable way, and to facilitate the creation of new math font sets. The current situation makes it impossible to use non-TEX math fonts from e.g. mathematica without *lots* of vf trickery.

These things are all very easy to fix in the executable, but it won't do any good at the moment, because we are still stuck with the TFM format.

16. *The way TFM and VF formats are defined and implemented is the primary cause of the current font chaos*

If we want to adopt a new format, the extensibility of the syntax of PL files is to our advantage, even allowing new features to be added in the future while remaining backward-compatible. But, although there no longer is a real reason for binary file input as speed or disk space optimization, binary files *do* have the advantage of being non-editable (meaning that the chances of a user accidently breaking them is very small).

17. *We need symbolic names for characters*

TEX currently uses encoding instead of glyph names. Encoding is old-fashioned and merely a speed optimizing thing. The coupling of glyph-name – character should be a TEX internal operation.

The used named characters from the fonts should be deductible from the output (DVI) file, to prevent reencoding issues in postprocessing applications. To reach this goal, it is very likely that TEX needs an internal naming scheme for glyphs that does not depend on font encoding. Work in this area is already being done by the eTEX team. It is considered unlikely that using unicode will solve the problem, but it might well be that a solution based on the predefined set of unicode names (the road taken by Omega) is the right way to go.

18. *Ligatures and kern info should be independent of the character metrics*

Ligatures can be present in the current font definitions, but we would like to be able to modify the lig-table internally from within TEX. This request has already be passed on to the eTEX group, but it needs a more general solution than the primitives that were proposed to eTEX(`\noligs` and `\nolig`<*char*>). Likewise for the kerning tables.

The mechanism by which a user loads fonts into TEX's memory is much too simple. It should be possible to specify encodings, kerning info and ligature tables separate from the actual glyph dimensions. The ligature problem actually comprises two very different problems.

The simple case is most noticeable in typesetting verbatim stuff in non-tt fonts, something that is often needed for textbooks on programming languages.

The hard case comes from the fact that ligatures depend on the language, not on the used font itself. The spanish quotation e.g. is never needed outside of Spain, and we are all stuck with it now. Ideally, every language should have it's own ligature table, that is part of the language attributes just like `\patterns` are.

19. *Metafont is becoming outdated, even if TEX itself isn't*

A new version of metafont is needed that can generate acceptable outline fonts instead of the now used `.pk` format, and the use of non-metafont fonts (Postscript, TrueType) should be simplified. As stated in a previous article, TEX should take care of the virtuality of fonts itself. But that does not *have* to imply using `.vf` files. There are some other possible solutions that may not be as powerful as `.vf`, but are a lot less confusing: The only widely used applications of virtual fonts are reencoding and creation of composite characters.

User interface

Currently, TEX shows a weird duality: while mostly a batch tool, there are still a number of places where user intervention is needed.

On one side, if TEX wants to survive as a batch tool (either as a stand-alone typesetter or as back-end for e.g. SGML processing systems), it will need extensions so that it is 100% safe to run the program unattended. Thinks like breaking math formulas and placement of figures cannot be left to TEX on its own.

On the other end of the spectrum, TEX needs a real-time graphical user interface to satisfy interactive users (maybe this can be a partial implementation, like having GUI-based equation- or table-editors). This goal can only be reached if the GUI-based tools have fool-proof TEX input format that they can rely on.
There are two probable roads we envisage:

- Moving a large number of current macros into the executable itself will avoid confusion of macro formats, but there are still problems to be solved relating to redefined primitives.
- Allowing a tokenized input in a precompiled format would probably be better since it circumvents these problems. The idea is that, assuming we are an external program that tries to generate TEX code, we want to be very sure that `\par` really means `\par`.

But there are some other idiosyncracies in TEX's language that needs to be dealt with as well. Sometimes optional, sometimes not optional keywords and characters like equal signs; arguments with braces versus arguments that are space-delimited; confusing rules for spaces; etc.

20. *At all events, the language should be cleaned up drastically.*

The syntax should definitely be cleaned out. Anybody who has ever tried to write a non-trivial macro will know that even if your approach in itself is correct, chances are that the macro still won't work, because of a stupid mistake with `\expandafter` or extra/too few spaces. Solutions that use markup in the style of SGML or lisp would be vastly preferable over the current situation. The current syntax often justifies the following statement:

21. *TEX's macro language encourages writing garbage*

We can safely say that many sources look awful in terms of formatting, just take a look at the sources of the style used to typeset this article. (Or look at the sources of the TEXbook: the output is beautiful, the input just ugly.) In the hands of common users, bad input becomes bad output.

22. *We would profit from better programming primitives*

Finally, experience shows that format files are never simple and small, like Knuth presumed they would be. Instead, format files are complex programs with numerous interactions between the various parts. TEX's macro language was never supposed to support this, and as a result has virtually no programming support. Among the missing things are data structures like lists and queues; name spaces; control structures (like cases and while loops); signals; and reliable `\if` tests.