Hans Hagen

Hans Hagen is the principal author and developer of ConTEXt (`http://www.pragma-ade.com`), past president of NTG (`http://www.ntg.nl`), and active in many other areas of the TEX community.

[Interview completed 2 March 2006.]

Dave Walden, interviewer: Please tell me a bit about your personal history independent of TEX.

Hans Hagen, interviewee: I live in Hasselt in a house situated in the middle of the town, on a kind of dike, looking out over the Zwarte Water (Black Water). It's a rather large building (former bank, 52.35.20 N 6.05.30 E) of which Pragma uses a few rooms, including the former safe rooms which we use for the server farm. I did a lot of rebuilding myself (a hobby) and made sure that the infrastructure was right (and that I had enough room to store my collection of books — another hobby).

My educational background is: gymnasium with a science focus followed by a few years physics and finally educational technology (at that time a nice mixture of exact and social sciences). Being among the first group of students doing this new direction, I had access to rather modern VAX/VMS computer equipment but TEX was not part of the game. We had to write many papers and used simple ASCII editors for that; the output was printed by high speed Daisywheel printers. I had a number of part time jobs in the academic department which gave me the opportunity to explore my programming skills (data analysis and such). Since we didn't have formatters, I wrote a pagination program (in Pascal) that acted on ASCII input. The program was used by other students as well.

After graduation in 1986, three other people from the same year and I started Pragma which focused on consultancy and development of learning materials, mostly for companies. At that moment we used personal computers in combination with our own editor and pagination software (faster and more efficient than what was available on the market... it was the MS-DOS era, before Word Perfect and Word showed up).

DW: How did you get involved in TEX and come to develop what is now called ConTEXt?

HH: At that time I knew TEX only from *The TEXbook*, which I had bought because it looked intriguing. At a certain moment we ran into a project where complex math was used, and I decided to give TEX a try. We bought MicroTEX and started playing with LATEX. We had to hack our way around in order to get rid of the funny chapter openings and such. Fortunately the US hyphenation patterns gave acceptable results for Dutch. Typesetting was slow and printing even slower, so it gave us an excuse to get a bigger PC and to buy a high-end printer (one of the first fast duplex laser printers). Configurability of LATEX was zero which was rather frustrating. After a few chapters we had managed to upset the customer, and the project was canceled; they were kind of offended that we had redone the typesetting which they had (proudly) done with a multi-head typewriter.

We kept using TEX, but with a low profile. We moved on to PCTEX and spent some money on buying updates for LATEX, Dutch hyphenation patterns, manuals, etc. After the first update of LATEX, I found out that I had to redefine the structuring commands again (to get rid of English labels), and so we started wondering about a more configurable setup. I started looking into other systems (Arbortext, 3b2, Berthold, Interleaf and the like), but in the end I decided to stick to TEX. I ordered LAMSTEX and INRSTEX, TaBlE and PiCTEX, and for a while we used those (I've never understood why LAMSTEX didn't replace the LATEX of those times). I found out that I could configure INRSTEX quite well (it had better hooks than LATEX for customization without digging deeply into the `.sty` file itself). I started writing wrappers around the code; by "wrappers" I mean macros that provide particular functions while hiding the code that implements those functions. Actually, we already used wrappers around LATEX code, most noticeably to provide higher levels of abstraction for itemization, for example:

```
\startitemize[n] : numbered
\startitemize[a] : characters
\startitemize[1] : symbol ’1’ (bullet)
```

The more I understood TEX, the more pieces of INRSTEX we replaced, and after a while we realized that we had a kind of new macro package.

Also, if I hadn't been able to implement proper graphic support by then (none of the TEX packages we'd run into provided what we needed), we'd probably have abandoned TEX. We needed all kinds of graphic formats, scaling, and more placement options; normally graphic inclusion is controlled by specials. Such things are very driver dependent, so a `\special` driver system was one of the first things that we made; nowadays there are all kind of packages for that but not when we were beginning. Also, we were not "on the net" and didn't have email (both were rather academic at that time), so we were on our own. I had to (re)invent wheels. -)

What evolved was a system that, like LATEX, provided a set of macros for coding a document; both structuring and layout definitions were part of it.

DW: When did you begin to call it ConTEXt, and why did you choose that name?

HH: Somewhere around 1996, I think. It was pragmatex before that. Context = con tex t — "text with tex".

DW: Please continue your description of its evolution.

HH: Right from the start it was keyword driven which in the end made us run into TEX's memory limitations. We got around that, and as a side effect we were able to provide a multi-lingual user interface. (Originally, ConTEXt had a Dutch user interface.) Implementation of a multi-lingual user interface was triggered by the fact that there was demand for our TEX macros by LATEX users of an English version of a chemical typesetting package that we had written. All our development is rather user driven (by my own needs, my colleagues' needs and, in the last couple of years, users' needs). Fortunately the design of the system makes extending it easy. The time involved in writing macros mostly goes into figuring out how best to interface that feature, how to let TEX deal with it (TEX has its limitations), and how to make sure that it does not interfere with existing code.

The development was purely driven by demand and configurability, and this meant that we could optimize most workflows that involved text editing. I wrote a more integrated editing environment (in Modula 2) which enabled us to structure projects in such a way that we could produce multiple outputs from one source. This is where some of the reusability features come from. Writing the macros took time, but we could update documents and (progress) reports so quickly that it paid off. We could offer authors high quality output immediately. Another driving force was the lack of communication: we were not aware of the user groups, and we were isolated from the net (not being involved at a university). By the time that we started visiting meetings and getting TEX CDs, ConTEXt was well on its way.

The tools that we used grew with the macros — faster machines, bigger TEX's (most noticeably emTEX). We bought a high end, high resolution, and high speed Océ laser printer which was probably the only printer put in the market as a DVI printer; but, at the time we got it, it only understood PostScript. The only way to achieve the quality we wanted was to use dvipsone and outline fonts. Years later, by the time we moved to pdfTEX, we had traded the machine for an even better one. The reason that I mention this is that, with the improving quality of the printers, we also set higher standards: there is a big difference between the 300 dpi bitmaps (a rather bold look and feel) and high res outlines. It's the high quality that got us hooked to TEX. Nowadays I cannot imagine using TEX on a low resolution screen either, especially since we started using MetaPost graphics.

Around 1996 ConTEXt started being used outside Pragma. The first users were Taco Hoekwater, Berend de Boer, and Gilbert van den Dobbelsteen; the first international user was Tobias Burnus who did a lot of testing. Around that time the company started shifting its focus to more advanced typesetting jobs (complex manuals, interactive documents). We wrote specialized environments, for instance for cross-linked dictionaries that we needed in projects. Currently we mostly work for educational publishers (PDF-to-PDF and XML-to-PDF workflows) and we make specialized styles for companies who use ConTEXt for typesetting purposes.

DW: I gather that in addition to the basic pdfTEX engine, you also use MetaPost. What other major subsystems like this do you use in your collection of tools, and can you say a few words about how you integrate their use?

HH: For a long time now, MetaPost support has been tightly integrated in ConTEXt which means that one can make graphics that adapt to specific situations (layouts, structure, etc.).

Because TEX lacks a command line interface, the preferred way to run ConTEXt is to use `texexec`. Originally this was a Modula program, later it became a portable Perl script, and nowadays it's a Ruby script. Because TEX jobs normally involve multiple runs, `texexec` can keep track of how many runs are needed; there is one utility file where all cross references, indexes (there can be multiple), lists (there can be many), and other information goes; index sorting is done by `texutil`, which is now integrated into the Ruby version of `texexec`.

The `texexec` program can take care of making formats, running jobs, efficiently running subjobs (like MetaPost) when needed, etc.; there are also provisions for runtime graphic conversions and the like.

ConTEXt comes with many additional tools, most notably `texmfstart`, which can be used to launch programs in the `texmf` tree; most of those tools have names like `textools`, `pdftools`, `tmftools`, `ctxtools`, `rlxtools`, etc.

Contrary to plain TEX and LATEX, ConTEXt comes with its macros preloaded and there is no distinction between backends, so there is no "pdfConTEXt" or the like. There are different versions of ConTEXt, but only with respect to the user interfaces (so in a French user interface, one uses French commands and keywords).

Sometimes there is a tendency in the TEX community to "put things in corners" (as we say in Dutch — "pigeonhole" in American English): if I show examples of using ConTEXt for educational documents, then people think ConTEXt is meant for educational stuff; or, for some time, I was always one of the first to show advanced interactive trickery and, thus, ConTEXt specialized in interactive documents. When I was sitting in on one of the excellent ConTEXt tutorials by Günter Partosch, he once remarked that "it took him a while to realize that ConTEXt was a complete system". So, even if you do presentations, write articles, etc., it takes a while for people to see the whole picture. I found that quite interesting to observe. Less fun are remarks such as "why do you use TEX if you don't need math?", but nowadays I seldom hear that.

DW: How big is Pragma now?

HH: Two of the other three original founders later dropped out, another got associated, and so now we're two in Pragma-ADE (`www.pragma-ade.com`) and three in Pragma-POD (`www.pragma-pod.com`).

DW: Do you have designers on staff, or do you mostly implement what your customers want?

HH: We don't have designers. I design the documents that we make here. For publishers, in most cases we get a design made in a DTP program and then turn that into a style. The biggest problem there is that such designs are seldom fully consistent (which may make mapping messy), there is no real systematic font and color usage (so one cannot make a nice systematic set of definitions), and the tests are done with unrealistic samples (so that one ends up with patch upon patch). Where possible we try to use ConTEXt core functionality. When implementing a style, most effort goes into the many kinds of graphic placements and in layout-dependent graphics. Also, grid-based designs can have their own dark corners. I'm pretty sure that most users don't run into those things and therefore may wonder why some features are supported in ConTEXt.

By the way, a good example of a design that could be implemented quite well is the journal of the Dutch math society, NAW magazine (`http://www.math.leidenuniv.nl/~naw/serie5/index.php`), but there the designer knew that a systematic design was needed which resulted in a very diverse and interesting design (many sub-layouts).

DW: You mentioned supporting ConTEXt users. Is there a distinct ConTEXt discussion list?

HH: There is a quite active ConTEXt user mailing list: `http://www.ntg.nl/mailman/listinfo/ntg-context`. There is also a development list, a bug/feature tracker, and a very actively maintained wiki called the ConTEXt Garden (`http://contextgarden.net`). The wiki project is led by Patrick Gundlach and sponsored by DANTE.

DW: To what extent are you involved in TUG, the Dutch TEX users group, or other TEX user groups or other more or less formal TEX "institutional" activities?

HH: I'm just an ordinary member of TUG, go to conferences, occasionally provide an article, and participate in program committees when asked. I'm currently the president of NTG, the Dutch-speaking user group. We have some 300+ members, two meetings per year, and a magazine. We are financially healthy which means that we can participate in projects that need funding. Regarding other user groups, I try to attend GUST (Poland) and DANTE (Germany) meetings since they are, apart from being TEX meetings, also meetings of TEX friends. I have been to other user group meetings as well.

There is considerable informal contact between the presidents of TUG (Karl Berry), DANTE (Volker Schaa), GUST (Jerzy Ludwichowski) and NTG (me). All of us are involved in projects that involve multiple local user groups, and we try to keep each other informed about projects, conference schedules, and issues that concern the TEX community.

DW: There is frequently talk in the TEX world about the limitations of LATEX, the need for improvements to TEX, and sometimes about what a "mess" the underlying TEX engine is (both fundamentally and with the enormous patch files that are used to modify and extend the basic TEX engine). You have taken matters into your own hands regarding making TEX work for you (and other users who see benefit in the things you have implemented). What do you anticipate as the future of your development efforts: more incremental developments and additions to your existing set of tools as new problems need to be addressed, or something dramatically new? And why?

HH: I wonder if I should comment on the limitations of LATEX, because I don't use LATEX. Users should use what they like best. What I do remember is that writing styles and extensions involved hacking around in the kernel. I believe that extensibility has never been part of the concept and that shows. I also know from talking to the core LATEX people that it's very hard to improve things once users start doing that kind of hacking. Also, the output needs to be "")as it was before), a restriction that I didn't put upon myself with ConTEXt. When solving a bug gives better output, so be it; patterns change, fonts change, and on the average no user will notice differences. In practice ConTEXt output is rather stable and does not change, unless you change your settings.

Anyhow, the LATEX people are forced to operate within far more strict boundary conditions than the ConTEXt people. Also, I found ConTEXt users to be very demanding but realistic: if they want some feature, they send examples and are willing to test. ConTEXt users have no problem with updating frequently because some new feature will suit their needs. Because the documentation lags behind, users go into the source code (where they often can find examples), and so far this has not led to users hacking around and making kernel-dependent extensions. There are well defined hooks for that, and users know when to ask for a new hook.

Regarding the basic TEX engine, I agree that things could be improved. TEX is written for a "write a style per book or series" kind of workflow. Because not everyone is willing to program, macro packages fill in a gap. One can reuse code, more easily configure the output, etc. When writing more complex macro packages, one longs for a language that has a bit more of the features found in modern scripting languages, like Ruby. (By the way, a future version of pdfTEX will have an embedded scripting engine, Lua, but that's another story.) It takes a while to get a grip on the TEX language and more time to get a feeling of what TEX is doing. Because most TEX code that is written (especially when computers were more limited) is quite unreadable, I never spend much time on looking into other macro writers' code (apart from INRSTEX). I learned it the hard way and at a certain moment entered a stage where one could "think in TEX". Every now and then I spend some time cleaning up the code. We have more memory now, so we can write more verbose code. Sometimes I keep old code around, just to show the stepwise improvements.

Should TEX be extended, and in what way? I often wish that TEX could give me more information about the state it's in. We can use a few more features as well, but not all wishes fit well into the way TEX operates. For instance, writing a decent multi-column routine (with advanced float support) is non-trivial. If one does not want users to clutter up their document source too much (one of the ConTEXt principles is that users should never have skips in their documents), one for instance needs the ability to set the height and depths of top and bottom lines in boxes (now a pdfTEX feature). One needs proper list processing (non-interfering whatsit nodes), more control over inserts, etc. However, whenever I look into this, I realize that Don Knuth stopped putting more features in TEX at the moment when the solution space became too fuzzy. Take for instance grid snapping (our customers demand that). In desktop publishing one can manually arrange things on the grid but what to do in a automated system: does a graphic's caption need to be snapped on the grid; with multi-line captions in a smaller font, is the first line snapped or the last line; what if there are several graphics stacked; what is the expected size of a graphic and what does the top of a photo align with? No specs, no solutions. ConTEXt can handle these things quite well; but, apart from some support from the TEX engine, I cannot imagine a generic model that will fit all macro packages equally well. This means that one will find alternatives in ConTEXt: multiple table mechanisms, more than one multi-column routine, and (more hidden) knobs to change the heuristics.

A good example of extensibility is ε-TEX. It provides a `\protected` definition prefix. Both ConTEXt and LATEX can use that, but if you look at the low-level already existing `\protect` macros, you will notice that they are needed in completely different situations. ConTEXt needs protection in its key value parser while, if I remember right, LATEX needs protection when one writes to the table of contents and such. Another example is `\dimexpr`. Instead of a clean expression engine, we have something that is modeled after a macro package (I'm told) which makes it less useful than had it been very generic. Or take `\scantokens`, which has some limitations built in that make it difficult to use in the situations where ConTEXt could benefit from it. So, each macro package has its needs, and it's hard to satisfy all. It's also hard to convince implementors that you need a feature that they themselves see no need for. Now that ConTEXt has become more widely known and accepted, things get easier. Also, pdfTEX now has become a place to add new features which happens on a regular basis.

The future is hard to predict. We're lucky that we didn't announce ConTEXt 3. I think that we can safely say that we now have the ConTEXt 2 code base, not that users will notice. The major change in the last couple of years was the (mostly automated) conversion from low-level Dutch to low-level English, so that more developers can participate. The formal user interface has always had a formal definition (nowadays in XML) so as long as this file is in sync with the code base, manuals that use the automated syntax charts will be adapted automatically. Not dramatically, but also not without impact, has been the integration of MetaPost. I think that it resulted in more MetaPost users and more interesting documents. Things like the evolution of color support (more color spaces, separation, etc.) are not that much of interest to users (apart from those active in high-end publishing).

For practical reasons we will skip version 3 of ConTEXt, and move on to version 4. That version will use the Lua scripting engine for certain tasks but it's hard to foresee to what extent. Who could have foreseen that adding MetaPost support would have resulted in arbitrary (nested) backgrounds behind the text flow? We will keep cleaning up the code, provide more support for Aleph-related features (bi-directionality), support OpenType, and keep exploring PDF features. So far I have always able to do whatever I wanted with TEX and ConTEXt, but that may change when we really hit the limits. In that respect it's interesting that most of our projects demand rather advanced styles, but seldom high quality typesetting (no hyphenations, ragged right, inconsistent font and color usage). One of our current projects involves manipulating PDF files (we split files, reassemble them, parse code and renumber chapters, sections, graphics and whatever, add fonts and other resources) while it would be more trivial to generate the files directly from (for instance) XML sources. Here ConTEXt is just an advanced reassembling tool that, in the process, adds some pages. So, who knows what version 4 will bring. I'm pretty sure that there is already much more in ConTEXt than the average user realizes.

When we talk about ConTEXt, we are also talking about the scripts. These are becoming more and more advanced, a process which is driven by the ways we use ConTEXt in more complex workflows.

One area where ConTEXt will expand is fonts: we now have two complementary font mechanisms; and, as soon as some restrictions from TEX are removed, I expect more advanced (OpenType) features to become available through the user interface. Another thought that I'm playing with is the ability to make spin-off macro packages: smaller, less and/or more directed features, for special purpose usage.

DW: Major advances in the world of TEX often seem to be the work of a key person who is largely self-directed and perhaps a few helpers (e.g., TEX originally, LATEX with Lamport originally and Mittelbach later, you with ConTEXt, Hàn Thế Thành with pdfTEX, etc.). The user groups don't seem to have the financial resources to undertake major advances, and them commissioning people to do such work doesn't seem to work so well anyway. What do you think the few best things are that the user groups can do to have significant impact on the future viability of TEX?

HH: Financing is not really needed for TEX to get developed, unless you know what you're spending the money on. Good examples are the font projects: there is a clear goal, we know that the people involved can do it, and the budgets can be kind of fixed. For example, recently the user groups decided to fund the integration of the free fonts that come with TEX and to make OpenType versions; such a project will cost between 30000 and 50000 euro (especially if we also deal with math). I'm one of the initiators, so in a sense responsible for spending quite a bit of user group money; but knowing the people involved, I trust that the work will get done (as long as we don't discuss the project to death). One reason to fund this project is to allow the people who are involved to spend substantial time in a small time-span; also, the project needs to be done fast, because we need (free) OpenType fonts in order to be able to let pdfTEX go OpenType. No OpenType support means the end-of-TEX, just as no PDF output would have meant the end-of-TEX some time ago. If TEX cannot be used in high-end publishing environments, it will end up in the margin.

Apart from small scale funding, I think that putting money into TEX development (a replacement) is not needed and counterproductive. There has been an attempt (the NTS project), but this also demonstrated that something really new is also politically tricky. It divides the TEX community; one ends up in discussions about programming languages, what's best and who better and... Don Knuth once said that the developer, first user, and writer of the first manual should be one person, and I think that he's right. If I wasn't using ConTEXt (in all its aspects) myself, it would not be as it is now. As an example, look at XƎTEX, an extended Mac version of TEX. It's done by someone who works in an environment where it's used. It evolves in an environment of demanding users. Being a Mac thing, it is no surprise that the focus is on fonts.

When there are talks about a radically new implementation, my first thought is always "nice". The problems that we're faced with that are hard to solve in TEX do not necessarily have a well defined solution space. This is why many efforts end up in interesting (in themselves) discussions, but no solutions. Also, I'm pretty sure that the solutions that I'd like are not the same as those that others want or need.

pdfTEX is a nice example: Thành just did it, used it himself, and now we have something that simply works. Of course, there are now more people involved in the development, but it's Thành who got it rolling. Some people joined in, used it, and gave feedback, but I'm pretty sure that a "program by committee" would have led to nothing.

One of my favourite examples is positional information. With TEX there has always been a lot of talk about not knowing the position of something (for example, a bit of text) on a page. It does not really fit in a system where the flow drives the makeup (breaking paragraphs and pages) more than manual positioning. Discussions quickly lead to "it cannot be done" and "if we should do it". So, at a certain point I asked Thành for a way to tag positions and write out those positions after a page was done, and within a day I got an experimental feature. It was after a while that I realized that this feature had been available in TEX right from the beginning: use specials, postprocess the DVI file, and use the relevant information in the next run. As a result I could convince the dvipdfmx author to provide this positional capability, too. For twenty years this solution had been sitting there and kind of went unseen because discussions made it too complicated.

Another example is line numbering, which would be a piece of cake if we had an `\everyline` feature. If we simply accept that the line-based content is not part of the paragraph but is added after the lines are broken, I can imagine a solution (it's a bit similar to the output routine). On the other hand, if we keep thinking of a feature that interacts with the paragraph, no solution is visible. TEX implements generic mechanisms, but sometimes a less complete solution is okay for what we want to achieve. For instance, think about multi-directional typesetting — not all directions have the same demands; or compare languages — Chinese line breaks are based on the surroundings of a character, and so we can make do with a more limited paragraph builder but with different characteristics. This is why an incredibly simple positional system can be so powerful: macros can do the rest and the author knows what he's doing and what cannot be done.

The best things the user groups can do is to organize meetings and collect people. DANTE and GUST are great examples of that. DANTE has spent much money getting people together to discuss distributions. User groups can also sponsor (small scale) meetings of developers. When I run into someone who I think should attend a user group meeting, I try to get him or her there; money on that is well spent, because once such a person knows the crowd, he or she may continue to participate. Diversity in spending money is better than focussing on a few projects. Therefore, the user groups can best spend their money on making conferences cheap and collecting people. That will keep the candle burning.

One thing that's also needed on the long run is a solid TEX code base. It is sad to see that the Unix and Windows development trees are not in sync (combined). When Fabrice Popineau (who had to merge his changes into a copy of the code base each time an update was made) decided to quit participating in TEX Live (and maintain an independent tree, just as the MiKTEX people do), I wonder how many people realized that the TEX development community had failed badly. We see big and complex programs being made available on the major platforms, but somehow the TEX world does not manage to get that done in an easy way. We cannot afford to lose people like Fabrice that way.

DW: A final question: what are your ambitions for your company and ConTEXt going forward?

HH: The future of Pragma is hard to predict. I hope that we can keep making a living the way we do now. I also realize that a small company like ours has limited possibilities, and supporting ConTEXt (reading mailing lists, fulfilling user requests, attending meetings, participating in user groups) takes quite some time. One problem our company faces it that we always get the kind of tricky projects (time consuming, imperfect resources, etc.), while you sometimes need a few easy going ones, simply because we then can have some return on investment on the tools we made. TEX-based solutions scale incredibly well, and it would be nice to have more projects that scale to the extreme. There are numerous possibilities for applying TEX, and I wish we had a few projects that we could "do on our spine" as we say here. Unfortunately TEX is not always seen as part of a solution. Anyway, we have been around for a long time, and we will not go away easily.

As for ConTEXt, the user base is growing, and I see users doing very interesting things. TEX still draws new (and young) users and that is good. As long as I have needs, and when users have requests, ConTEXt will be extended. Of course I have some ideas about new directions, but I don't want to make false promises. Lately, users have started making their own modules, writing their own manuals (either using or not using the ConTEXt "MyWay magazine" style). And what is important for me is that there is a growing group of users who know the source code and can contribute. The time when ConTEXt was a mostly one-person job is behind us. For instance, Taco Hoekwater is now the driving force behind releases and the subversion repository (dating back to 1996). He's the best TEX-analyst (program and macro code) I know, and without him ConTEXt would not be where it is now. We have users like Adam Lindsay and Henning Hraban Ramm who are very active with fonts, and by now Mojca Miklavec is a real encoding guru. I could mention more names, but I'd say, take a look at the wiki and see who contributes. Take some time to look at the wiki anyway, and you'll see what wonders Patrick Gundlach did with the source browser and quick references browser. Long ago, a user told me that one of the charms of ConTEXt is that there is a short distance between the users and the core developers, and I hope that we can keep it that way.

DW: Thank you, Hans, for taking the time to participate in this interview. I find your ideas to be highly insightful and stimulating. I will have to give ConTEXt a try with the book I will start work on in the near future. I also hope I meet you in person at a future TEX conference.