The State of ConTEXt

Hans Hagen
Pragma ADE, The Netherlands
pragma@wxs.nl

Abstract

In this article I will describe the current state of the ConTEXt macro package and the forces that play a role in its evolution. I will also indicate the directions in which we look for further developments.

1 ConTEXt developments

The public part of the ConTEXt story started around 1995. If we summarize the main developments in this macro package we can roughly identify the following points of focus:

- a configurable environment where users can define styles, using an interface in a language of choice; the multilingual interface was first needed when the chemical package PPCHTEX was adapted to English and made generic
- support for document collections such as we find in educational settings, with a focus on re-usability; multilingual support, selective processing and dedicated modules for chemical formulas and consistent usage of physical units evolved from there
- features aimed at highly interactive documents, optimized for reading on computer screens; support for one source, multiple output was part of that
- extensive support for grid snapping combined with advanced multi-column typesetting
- typescripts as a means of building font collections and combining typefaces in many (possibly weird) ways
- all kinds of fuzzy configuration options needed in order to mimic the behavior of desktop publishing applications
- integrated support for processing XML documents and using XML databases

Extending and improving ConTEXt have never been related to strong versioning or promises for successors. Part of the game is that we try to remain downward compatible. And so, officially, we still have ConTEXt version 1. Successive releases are tagged by date.

The most recent change was not so much related to new features but more to the machinery behind the screens. Those who have looked into the source code probably have noticed that for reasons we will not discuss now, keywords and variable names look rather Dutch, that is, until recently. Around August 2004 we made the move to a low level English interface. Although we had some help from a Perl script that had been written for this purpose years ago, still quite some manual checking had to be done.

This does not mean that ConTEXt is completely clean under the hood. When we started developing the system, TEX's were small, and so we ended up with quite some dirty (not that verbose) code. One can easily recognize the older code, but we hope to weed out the ugly bits in due time.

There is good reason to qualify the current version as ConTEXt version 2. The reason for this is that users who use low level Dutch keyword constants (prefixed by `\v!` and `\c!`) in their non-Dutch styles, now need to translate these into English. A bonus is that third party extensions will be easier to implement. Such developments will further be stimulated by Taco Hoekwater's ConTEXt API project and Patrick Gundlach's ConTEXt interface description project hosted at `contextgarden.net`. I must admit that the decision to go low-level-English now and not later, was triggered by their initiatives.

Of course one can legitimately ask whether there is still need for further developments in TEX macro packages like ConTEXt. At PRAGMA ADE we deal with documents coded in TEX as well as the more avant-garde XML format. It cannot be denied that XML coding makes documents much less error-prone: it's much simpler to check the syntax

Editor's Note: This article is reprinted from MAPS 31 by kind permission of the author and editor. The author made several related presentations at the Practical TEX 2004 conference.

of an XML file than of a TEX file. However, it can also not be denied that the loss of typographic detail (or more precisely: the means of authors to improve the look and feel of the final result) is a high price to pay. Of course there are also document types that cannot easily be covered by a manageable set of XML elements. Just think of highly complex math, physics or chemistry, or manuals that use a wide range of visualisations. One easily ends up with something that, although coded without backslashes, looks rather familiar to TEX users.

An even more important observation is that whatever means of going from document source to typeset product we choose, the visualisation problem will not change. No matter how many tools (or macros) one writes, differences in designs (and not seldom inconsistencies in designs) demand unique solutions. Although one can easily become overwhelmed by the possibilities that today's publishing tools provide, there is still a place for the proven TEX technology.

2 ConTEXt and browsers

Recently we redesigned the PRAGMA ADE web-site, a site that is mostly dedicated to ConTEXt. The HTML pages are generated from XML sources using XSLTPROC. Some of the PDF documents are generated from the same XML code. In addition, we generate templates and interfaces for the EXAMPLE framework, of which we now run an instance on the web site. This framework is a shell around TEX and friends, and provides features like page imposition and font tests, wrapped in an interface, but very TEXish underneath.

When rebuilding the web site, it was enlightening to find out that standards like CSS were not always precisely supported, even after being around for many years. Firefox (Mozilla Gecko engine) does a decent job. But for Internet Explorer, we have to cheat dimensions and use dirty tricks to get the alignment right. Opera was not that bad, but could not handle relative dimensions well. In the end we had to follow yet another approach to make Apple's Safari Browser (based on the KDE engine) happy as well. One lesson that I learned here was that even an abundance of implementations (or renderers) and tons of documentation (it's easy to find info in the web on CSS and HTML) makes defining a simple layout a painful and time consuming process. It's also interesting to see that the amount of XSLT code needed is not necessarily smaller than the ConTEXt code needed to generate similar output in PDF. Although the TEX community is under pressure of evolving techniques, it should also realize that its huge repository of tools and macros is not that bad after all.

Interestingly, browsers can handle complex operations, like displaying Arab or Chinese and handling widgets and JavaScript quite well, but setting up a simple geometry based on fractions (percentages of the screen size) goes beyond their capacity. Something similar can be observed with the XML related CSS cousin XSL-FO: I still have to run into a nicely typeset book done that way with a better than mediocre design. Again the focus seems to be more on the machinery around it, than on the creation of masterpieces. But then, this may well be beyond its purpose. Whatever a TEX user may think of CSS compared to his or her favorite macro package, its influence is undeniable. The evolution of the Mozilla platform demonstrates this: it provides a user interface builder based on CSS and XHTML called XUL. When PDF came around, I made some documents that could be considered to be programs. It looks like in the end typesetting and user interfacing finally meet each other.

3 Future developments

The majority of documents is a collection of paragraphs of running text, itemized lists, a few graphics here and there, and a couple of tables. TEX and TEX-related packages can handle such documents with ease. However, it seems that even in automated work-flows, where most of the interface can be hidden, TEX is seldom considered to be an option. But, when no other alternative is available, or when other applications failed to perform, this 25 year old program can come to help. It's interesting to observe that the TEX community can still attract new users who don't consider the user interface too much of a problem. So it definitely makes sense to continue development, if only because there is still a large group of documents that demand such tools and typographical detail. As long as TEX can keep up, the ConTEXt story will continue and we will see version 4 (extremely modularized), version 8, 16 and maybe 32 some time in the future. In the end it may be that properly typeset documents where time and effort is put in the look and feel, become a niche, and make way for documents with a minimum of design that can be generated each time they are updated, using the user's preferences.

What is currently happening at the ConTEXt frontier? ConTEXt has been ε-TEX aware for a long time, and the PDFTEX engine is supported quite well. The good news is that PDFTEX is still under active development. For instance large parts of the font handling were redesigned, and paragraph optimization (PDFTEX implements a font expansion algorithm akin to the HZ micro-typography algorithm by Prof. Hermann Zapf) as well as protruding (hanging punctuation) have become more user friendly. ConTEXt supports both mechanisms quite well.

With the arrival of Aleph, the stable descendant of Omega, support for this extension will become more visible than it was so far. Although UTF is supported, as well as some specialized Chinese encodings, using Aleph will bring Unicode support in the broadest sense, given that adequate fonts and hyphenation patterns become available. In many aspects Omega is not as multilingual as advertised, and certainly not by nature. Omega and therefore Aleph provide some mechanisms, but one still needs macros on top of these to tie the directional typesetting to actual languages and layout. Taking fonts as a starting point, the Mac OS X specific unicoded TEX variant XeTEX also looks promising. According to one of the ConTEXt Mac OS X experts, Adam Lindsay, hardly any extensions to ConTEXt are needed in order to get documents typeset in virtually every script.

Other developments that may become of interest are Taco Hoekwater's merge of TEX and METAPOST. There ConTEXt will not only benefit from a speedup due to more efficient inter-process communication, but it may also open new worlds. The average user will probably not use ConTEXt the way we do, for instance to create DTP like output from XML sources, which often means multiple calls to METAPOST per page. Think of documents with 250–500 pages, hundreds of (possibly run time manipulated) graphics, thousands of calls to METAPOST, with an occasional size of over 500 megabytes, and you can imagine that any speed improvement counts. Most features that we use in projects end up in the kernel, and so many users may profit from an efficient integration.

I already mentioned XML. In the next couple of years, more ConTEXt subsystems will use this format in one way or another. If you take a closer look at the distribution, you will notice that quite some XML objects are present already, like in the figure database mechanism and other tools. New is FOXET, yet another XSL-FO engine. Formatting Objects (FO's) are a kind of building block to be handled by a typesetting engine.

Although FOXET ended up on our agenda due to some vague promises made long ago, the actual development of FOXET was triggered by the observation that the ConTEXt MathML engine is being used to fill in the gap in commercial engines. Why bother making small bitmaps (or PDF snippets) of formulas while TEX can do the whole thing? It is interesting to notice that most of the documents that this applies to are rather trivial to typeset with either ConTEXt built-in XML features or by using XSLT to generate intermediate TEX code. It is also interesting to observe that there are ConTEXt users who use XML documents with ConTEXt as a backend, thereby hiding TEX completely.

The magic sound of XSL-FO occasionally makes our customers express the wish for an engine that can handle them (even if their designs are not that well suited for it). Somehow the magic obscures the fact that it's a relatively slow process, that it may take longer to implement (as said before: the problem does not change), does not necessarily lead to well typeset documents, et cetera. If one knows that something is possible (and with TEX much is possible) the demands of designers are seldom adapted. When something is not possible at all (and this occurs with XSL-FO) my guess is that the demands will be dropped. Float handling and marginal notes are examples of areas where TEX is hard to beat.

4 Paragraph building

So what about TEX's superior paragraph builder? Unfortunately most of the documents that we have to typeset professionally are designed by those who use DTP systems with poor quality paragraph builders. This means that they simply cannot believe that there are programs that can do a decent job. As a result we end up with colorful and abundantly illustrated documents that have rather complex layouts (especially if you take into account that they are typeset automatically) but with poorly typeset paragraphs, and that is what they recognize. It is hard to explain that by setting all TEX's penalties to their maximum, the solution space becomes pretty small. Even the somehow always demanded ragged right justification then looks plain bad. The problem for the TEX community is that alternatives for TEX don't have to provide TEX quality paragraph routines. As long as they can get the layout done, they win the game. ConTEXt users who like to look into the source will have noticed that quite some control was added in order to meet these demands, even to the extent that it may lower the quality.

So what good is it for TEX users? As with many things, it's no bad idea to take the best of all worlds. There is nothing wrong with DTP, and for many applications, an Office Suite does well. And for a certain range of documents XSL-FO is a good choice. Of course it remains puzzling why some of today's publishing on demand workflows are presented as something new, while in practice it already could have be done that way for decades using SGML and TEX, at far lower costs too. In some sense TEX was simply too far ahead.

One can mix those techniques. Just as one makes a graphic in a drawing program, one can imagine embedding a one page document coded in XSL-FO as a graphic in a TEX document. In this way we get a kind of 'placed XML'. And ConTEXt already can happily combine TEX and XML in such ways. Also, it's more convenient to store information in a standardized (XML) format, than to invent some syntax for each situation and develop different tools for each of them. For instance, if we want file information in our documents, we use `xmltools` to generate a directory database (this can be done at document processing time by using a system call) and we then let TEX filter the information from that database. Another example is OpenOffice. Anyone who has taken a closer look at this program will probably have recognized similarities with XSL-FO related developments. Seeing TEX as an alternative back-end for texts edited in that environment is not such a bad idea.

All these worlds can meet each other in ConTEXt. In ConTEXt, TEX and XML come together not only in FOXET, but also in what we've called 'The Example Framework'. The EXAMPLE logo has the x, m, and l hidden inside, but the actual purpose of this project is to hide TEX from users. On our web site you can play with some of these framework features.

It will be clear that the future of ConTEXt is to some extent related to the advance of XML, although the pure TEX approach will not be neglected. For many documents the TEX syntax (or in our case, the ConTEXt one) is quite well suited and efficient. Although I nowadays code most database related documents in XML (like the PDF showcase document interfaces) I have no plans to abandon TEX. Even thinking of coding a manual like the one about METAFUN in XML already gives reasons for nightmares. And so... plenty of ConTEXt ahead.