Does TEX have a future

Introduction

Making the transition from ConTEXt MkII to MkIV took a lot of time. In the process all kinds of code were evaluated, improved and, occasionally, removed. To some extent, the frozen state of MkII reflects the requirements of automated typesetting of the past two decades. Today, LuaTEX is advancing automated typesetting beyond what was previously possible. But do we really need it? In this article I will describe several issues we faced while rewriting the code, the choices, and compromises, we made. I will not attempt to answer the question whether TEX has a future, but merely offer you my own observations and thoughts.[1]

Media

It is not hard to extrapolate the advance of e-books, and the demise, in some countries, of paper books. Less demand for printed books means less need for typesetting. Of course, real-time rendering is also typesetting. But since there is no one format compatible with all e-book readers, publishers are unlikely to produce multiple device-specific versions. To what extent is a shift in the way documents are encoded important for TEX development? And as publishers cut quality and costs in an attempt to stay alive, who will want high quality output? Personally, I think more and more authors will turn to self-publishing. In this respect we might see a revival of TEX and more complex typesetting. It all depends on how important a particular look and feel is, and what price you are willing to pay to achieve it. Nevertheless, we cannot deny the fact that times are changing, and that technological developments will influence how TEX-like systems evolve.

From the start ConTEXt could produce very complex interactive documents. But apart from its inclusion in several projects, this functionality has never been in any serious demand by the publishing world. One reason for this is that compared to the printed product, interactivity is seen as an additional 'free' feature. As we enter the age of electronic books, we see that the features commonly used are only a portion of those available. Nevertheless, all this accumulated functionality is available in MkIV. When a typesetter has an eye for quality, interesting typographic and navigational details will appear.

Application

It is quite usual to find ConTEXt hidden in a larger publication workflow. In such cases the input comes from a database or some online editing environment. The layout, and therefore the typesetting, are often relatively simple. A predefined style tells ConTEXt how to transform input to output. The input may be predictable, but the user still has significant influence on the workflow. In this situation, what sets MkIV apart is its ability to analyze and manipulate data sets. MkII can also deal with data, but with Lua on board, MkIV solutions seem more natural. MkII is sufficient for traditional typesetting situations, but MkII is a dead end compared to MkIV.

We often talk of TEX users and user groups, but the more abstract term *usage* might be a better indicator of how much *TEX* is used. The number of TEX users is not growing, but TEX usage might be on the rise. Perhaps counting the number of pages produced with TEX is a better indicator of how prevalent TEX is rather than counting the number of installed TEX systems.

Coding

Another observation is that ConTEXt users often produce more advanced and demanding documents than I do as part of my work. For me, the biggest advantage of MkIV is its support for OpenType. Fonts are easier to install, and all those encodings disappear. Another advantage is that MkIV has a flexible xml processor built in, which can save you time in solving problems. Of course we continue to use and improve basic rendering capabilities, but we often have to simplify solutions when designers fail to see the possibilities of automated typesetting. Stability is often cited as the reason to use older combinations of TEX engines and macro packages. Ease of use and improved maintenance might be sufficient reasons to move on.

1. This text was copy-edited for Maps by Michael Guravage, whom I gratefully thank for helping me express my thoughts.

In the early days of ConTEXt my colleagues and I were its main users. One of the nice things with TEX compared to a word processor —never in my life have I had to use one—is that you can automate things. Imagine that you attend a series of meetings where several hundred learning objectives are identified, described, ordered and grouped. If you are in charge of such a task, it really helps to have a system where numbering and breaking pages comes for free. We were often able to get the adapted documents in the post within a few hours of leaving the meeting. The authors were impressed when, in the next stage of the project, we presented them with multiple professional looking documents derived from the same source. No other application could easily handle 500 floating images on 300 pages without crashing. This was the time that using TEX paid off for us. That was more than 15 years ago.

Such a TEX-based workflow is a sequence of edit, run and preview cycles; steps recognizable to any old-time computer user. However, it is not something that newcomers, like our children, might deem usable. It's not 'what you see is what you get', but the more abstract process of 'what you key is what gets done'. Wrapping TEX with a simpler interface would only hide its power, flexibility and charm. Then, you might as well use a word processor. Let's face it, using TEX directly only pays off when the user can separate coding from rendering, wants to have full control and desires to be independent of hard coded solutions. Try explaining that to a twittering facebooking kid. Regardless of how we move from MkII to MkIV, the route from source to result remains the same, and so does the intended audience. Updating TEX engines and macro packages will not increase TEX usage.

For some of our ConTEXt projects, traditional paper-based books are complemented by content intended for the web. Consequently, the document source is often xml. We could encode documents using a TEX-based coding, which, if they had the freedom to choose, would likely be more comfortable for authors to use. I wager that many authors who have used TEX directly still prefer it as an input language. Though xml is a widely accepted input and storage format, it is not ideal for typesetting. xml is geared toward publishing on the web and is not as expressive as TEX. However, reusing content is rare, so we needn't worry too much about encodings.

Coding in xml has some advantages for processing by TEX. There are no TEX commands for authors to misuse or redefine, and valid xml documents produce no errors. Another advantage is that styling and coding are completely separate. Of course, relieving the author of the responsibility of rendering complex documents can lead to sub-optimal output, unless the author is willing to adapt his content. The advent of xml has made people aware of the benefits of structure. ConTEXt tries to enforce structure, so TEX can fit nicely into modern publication workflows. However, for the quick and dirty one-time documents, the overhead of adding structure might not be worth the effort. So, even if in MkIV we promote using `\startchapter` over `\chapter` and `\startitem` over `\item`, we keep supporting the less demanding coding variants.

The styles I write today are a mixture of TEX, MetaPost and Lua. Solving the same problems with MkII, if possible, would require considerably more effort. Just as the faster Internet has become natural, so has the MkIV mix.

Double-sided

An electronic medium is single-sided. A book is always double-sided, and in the case of magazines and newspapers also multi-column. ConTEXt has quite some code to deal with double-sided layout. Sometimes TEX collects more content than can fit on one page. When this happens we have to keep track of where content should end up. For instance, dimensions and alignment conditions for margin notes must be swapped for odd and even pages.

In a single-sided universe, all the ConTEXt code that deals with `inner` and `outer` positioning and alignment could go away. Headers and footers could also be simplified. By removing the distinction between left and right pages, we could also drop some page synchronization code. Backgrounds wouldn't have to keep track of state either.

Related to this is page imposition. Page imposition is built into ConTEXt and is rather advanced. New imposition schemes occasionally appear through the effort of Willi Egger who not only typesets but also prints and binds books. The advent of a new folded paper gadget can be the impetus for adding yet another variable to control the position of pages.

Some of our projects require that we produce imposed products as part of an automated workflow. Cover pages, combined with back pages, are on the agenda for future integrated support. Since these features are applied to finalized pages implementing them is relatively easy, and they do not interfere much with existing code.

To separate content within electronic documents, we might end up with all sorts of cover-like pages. After all, additional e-pages are cheap, and color comes for free. This means that we might see more advanced page clustering and numbering schemes in ConTEXt MkIV. For instance, it might be nice if chapters had alternating or unique background colors. It would be even nicer if this property could be implemented without introducing new user commands in the source document.

Paper size

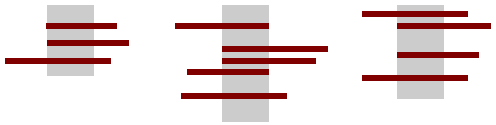
Paper books have standard page sizes; electronic books do not. Splitting tables with spans or large cells is somewhat painful. So why should we split a large table in an e-document when we could just as well scroll?

In some way we're going back in time. Long ago scrolls were used as a continuous medium. In that sense, scrolling on a display is not as new as it may look.

The concept of a page is derived from the medium — but what if we ignore this? For instance, if each chapter of a book were a separate entity, we could have one long page per chapter. This is problematic since TEX sets a limit on how high a page can be. But imagine that instead of thinking vertically we go horizontal. Headers and footers go away or get a new meaning, and the edges would give some indication of where we were. Perhaps we need a floating indicator; we've seen stranger things. Would this require a programmable viewer that we could control from our document, or could we anticipate standard features in viewers and viewing devices? Luckily for us we can adapt the TEX backend for either eventuality; at least we have done so for over three decades.

Floats

Floats are nice for paper. It is interesting to notice that in ConTEXt's early years floats were very prevalent in the documents we produced. In fact, they were a selling point. In educational documents especially, graphics need to appear near to where they are mentioned in the text. In a purely electronic document we don't need to struggle with fitting graphics on a page. Relaxing this requirement would simplify designs. Removing the corresponding ConTEXt code would definitely make the codebase leaner and meaner. But don't worry, we have no plans to delete anything.



What if we combine the previously mentioned vertical layout with horizontal extensions? Again with a finger we swipe our way down the page, where we run into an indicator denoting a larger image. Swiping our finger to the left displays the image; which might be accompanied by texts, images or animations. Another swipe and we're back in the main thread. It is amazing that we can do this with TEX. In fact we can proceed to multi-dimensional or even parallel documents. I remember turning the MetaFun manual into a QuickTime 360 movie. I must have a ConTEXt presentation style somewhere that implements this one page presentation where clicking on areas exposes different parts of the page. TEX is and will always be a fine playground for such concepts. MkIV with Lua and MetaPost makes it even finer.

Margins

The first step from a paper document to, for example, an e-book device is to get rid of margins. Due to technical limitations all devices shipped around 2012 have rather hard-coded physical margins. Perhaps one day we will have devices that have matte displays running from edge to edge. Imagine a device without buttons, logos or stickers proudly mentioning the internal chip sets or operating system.

The current tendency is to remove margins. In the near future we might see them coming back. Margins provide structure, and also room for various indicators and navigation aids. This is a good thing. Support for putting things in margins is quite important. In MkIV we already go further than in MkII and more will come.

Accessibility

A table of contents still makes sense in an electronic document, but what about an index? An index's usefulness is proportional to how carefully it was prepared. In many cases a search option works just as well. The concept of a table of contents can be expanded to include local tables and navigation aids that help the reader find what he wants. Similarly, we can collect information in multiple indexes. We added multiple interactive indexes to ConTEXt while involved in a project that produced quality assurance manuals. In another project we needed index entries arranged in a linked list, which is why this functionality exists in MkII. This cross-linked variant is not

yet available in MkIV – simply because I don't know anybody who needs it. Interestingly, implementing it in MkIV is far easier than in MkII.

A great deal of functionality, some of it even documented, is there because we once needed it. Take, for instance, flow charts. We can make really big ones. Selected cells can become hyperlinks — allowing us to jump through the document. Again, this functionality was a side effect of making those interactive QA manuals.

Mechanisms like these have always been part of ConTEXt, even when they make no sense for paper documents. They are more coding issues than demanding typographical challenges. They do not interfere with other typographical components, so simplifying or removing this functionality has no benefits. We can do much more in MkIV, but sometimes I get the feeling that less is more.

A lot of code in ConTEXt deals with structure. It makes sense to think about ways to improve how we gain access to it: linked lists, pop ups, summaries, reading routes, etc. MkII has several mechanisms that make controlled reading possible, but they never took off. In MkIV most mechanisms that structure data also retain part of it for re-use. Because we store data for use in a second or subsequent typesetting pass, information can be used multiple times.

Some mechanisms also support user data. For instance, when starting a chapter, besides setting its title, you can also name a variable that stores the name of an image — a sort of visual title. As this name is carried around, the image can appear as an icon in the table of contents and on the first page of the chapter. We needed this a long time ago in MkII. This is one reason why in MkIV we can now set user variables in commands that start chapters and sections.

In one project we participate in, a free math method, the content is first published on the web. Given the nature of electronic documents, it went unnoticed that, when typeset for the printed page, the document was quite large. Selective use of content, multiple products, and efficient typesetting are solutions to this. The e-book version is not constrained by the number of pages. Information can be repeated when needed; complemented with the necessary navigational aids. I'm confident that ConTEXt can deal with both variants.

There has been a time, probably due to the fact that I gave presentations showing pdf on a projector, that ConTEXt was promoted as a system for creating electronic documents in pdf format. This is just one feature, but interaction has always been integrated in the core — never an add-on. However, there is a fundamental difference between interaction in MkII and MkIV. Using different techniques in MkIV, we no longer have interfering status nodes. This makes the whole mechanism more robust, although internally it has become pretty complex.

Columns

Columns make sense in broadsheet newspapers and journals where one wants to put as much as possible on a page. But I wonder if columns make sense in electronic documents. After all, electronic pages are cheap, and getting rid of multi-columns makes typesetting much easier. In TEX the mechanisms that deal with columns, e.g., page builder, floats and notes, are often complex. The code can be pretty messy. It would be nice to get rid of this legacy.

A good application of columns can be found in parallel bible translations. Not only must the text be synchronized in multiple columns, it also has to be broken across pages in a reasonable way. Footnotes are another complication.

Will such products be made in the future? The production of printed encyclopedias has already stopped, and concordances might soon follow. On the other hand, the fact that Thomas Schmitz typesets sophisticated documents for tablets, notebooks, projectors, and paper indicates that, for critical editions, the future is not yet determined. And I know several TEXies who typeset catalogs for conferences and festivals where a proper paper version is the only way to provide an effective overview. All these documents share a mixture of one column, multi-column and specially composed pages.

ConTEXt currently has two mechanisms that deal with columns. The first mixes well with single column mode, the second is more powerful and encapsulated. In MkIV the pluggability of the output routine has been improved; so if needed we can support yet unforeseen page building schemes. Parallel streams are first on the agenda.

Move on

If we consider only paper documents, do we anticipate needing more typesetting functionality than we already have? Does it make sense to develop macro packages any further? Of course, it is not difficult to make a wish list including more support for complex critical editions and parallel typesetting of translations. For those who use a simple input format such as Markdown, existing ConTEXt functionality is more than sufficient. In fact, as long as we can deal with the concepts found in html we're okay. Most of these documents consist only of running text, tables, images, a bit of sectioning, itemized lists and maybe descriptions.

TEX is over thirty years old. It is still maintained and kept up-to-date. It provides users with a lot of freedom. It has an active user community. It is often chosen for long term use. It is boringly stable. With these attributes, we can safely assume that TEX will be around for a while. The same is true for macro packages. They will stay and evolve. But how will TEX change along the inexorable path from paper to electronic media? Typesetting habits change slowly, so we still have some time to ponder these questions.

On the other hand, look at how quickly the web is evolving, and how quickly younger generations adapt to new electronic devices. When using TEX it is natural to think in book-related categories. But just as a computer desktop is not a real desktop, an e-book is not a real book. Real books have a physical presence; we can hold them in our hands and turn each page as we read. E-books try to mimic these physical characteristics with ridiculous results. For example, you can choose an e-book from an e-book-shelf, and turning pages is simulated by showing the binding and a moving cut edge. But will we want or need these visual clues in the future when we have instant access to everything from anywhere on any device we choose? Why carry around a book when we can have its contents projected on our retina, or hear it spoken in our ear? Regardless of how TEX and its attendant macro packages evolve, we'd best refrain from predicting the future, let alone promote TEX as the ultimate and last word on typography. We can only hope that future hardware and software will allow us to TEX like we allow printers to use printing presses.

Hans Hagen