?

context

Rationale

Occasionally I run into a description of ConTEXt that contains observations that are somewhat off. It therefore makes sense to provide some insight in why this macro package looks the way it looks. What started out as a TEX only system evolved via adding METAPOST to the current hybrid system that also uses and provides Lua. However, the original goals and principles are still valid.

The system

The TEX macro language and interpreter are about automated typesetting. A collection of predefined macros is called a macro package and ConTEXt is such a package. The program that interprets the macros and converts input into typeset results is called the engine. An example of an engine is LuaTEX. The graphic companion of TEX is METAFONT, or nowadays METAPOST and ConTEXt integrates support for that graphical language and rendering. As ConTEXt comes with a bunch of tools that manage the process it is a sort of typographic ecosystem. The input can be a document encoded in structured TEX code or it can be xml. In fact anything that makes sense can be processed if only because we can use Lua to convert it runtime. The output is in most cases pdf, but xml (or xhtml with CSS) is also an option. In addition to static printable documents you can produce highly interactive documents for screen and reader.

The TEX system, and therefore ConTEXt, are quite capable to produce high quality output, can handle many scripts and languages, are able to typeset mathematics and can keep up pretty well with developments, thanks to the open character of the whole system. There is an active community that takes care of the software distributions and develops the engines as well as additional resources like fonts. User groups play an important role in this. The ConTEXt system is supported by active mailing lists and a wiki.

The syntax

The syntax is rather straightforward and uses commands that start with a \, the curly braces {} for delimiting variable content and the square brackets

[ ] for (optional) directives and settings. A good example is one of the oldest commands:

```
\startitemize[packed]
\item some text
\item some more text
\stopitemize
```

Before we ran into TEX we already had some programs that dealt with simple formatting and start/stop constructs were used there. The not delimited `\item` command was just taken from what we saw in other TEX code. In retrospect we should have used `\startitem` and `\stopitem` commands right from the start but it's only with MkIV that we start promoting this more strongly. Actually there are only a few commands that are not delimited and this is one of them. So, this is today's fashion:

```
\startitemize[packed]
\startitem some text \stopitem
\startitem some more text \stopitem
\stopitemize
```

One reason for getting rid of the few non delimited cases is that is't more cleaner in the perspective of hooking in code as well as exporting to for instance xml.

The (optional) argument between square brackets use those brackets because for a beginning macro writer (which I definitely was when I started with ConTEXt) it's not that hard to implement. It also looks nice on screen, especially when you use syntax highlighting. When ConTEXt showed up we still used our own editor and it used a color scheme that can still be found in the pretty printer code. So, to a large extent, the syntax has been determined by how well it could be visualized in an editor. We set up commands using:

```
\setupsomething
  [whatever]
  [key-1=value-1,
   key-2=value-2]
```

Spaces after a comma are ignored contrary to spaces before and after an equal sign. It takes more (and slower) code to do that and picking up keys and values is already slow enough, especially when we started doing it.

I'm writing this in the SciTE editor (2.29) using updated syntax highlighting code that was added to the distribution around the ConTEXt conference in 2011. Of course the TEX code gets highlighted, but so does METAPOST and Lua code. Runtime spell checking is included. Developments like that have some impact on the user interface and the way (low level) code is written. Visualization of code and source has been and always will be an important aspect of how ConTEXt evolves.

The multilingual interface

A second important aspect of ConTEXt is that there is just one version. I simply could not come up with a good reason to use different names for different engines. After all, most of the (older) core functionality is shared. It is always a surprise to run into descriptions of ConTEXt being dependent on pdfTEX, Perl, or Ruby. In ConTEXt there has always been an abstract layer between the core code and the backend and support and in MkII relatively little amount of backend specific code is loaded. In MkIV we have several backends running at the same time: pdf and xml export can run in parallel.

Being closely involved in the development of pdfTEX naturally means that I used that engine but originally we used dvipsone and ConTEXt MkII supports several backends. The dependency on Perl and later Ruby was only true for the script that manages a run (originally the index sorter was written in Modula2). One could run unmanaged but it's much more convenient for users not to worry about how many runs are needed. It also gave us the opportunity to provide command line arguments. And of course index sorting has to happen somewhere out of TEX itself so why not combine these tasks in one script. When we moved on to Lua it was natural to stick to only Lua and as LuaTEX is also a Lua interpreter, there is no extra dependency in ConTEXt MkIV.

We have only one version, but on your system you might find more than one `cont-*` file. The reason for this is that there are different user interfaces. We started with the Dutch interface and later added an English and German one. Of course some more followed.

When we started using LuaTEX, it quickly became clear that we had to split the code base and so we did. In fact one can now indeed claim a dependency: on LuaTEX. The fact that MkII is frozen is a clear signal that we see no future for the other engines in the ConTEXt community and their development is stalled anyway. The same is true for related technologies like fonts. Why stick to obsolete font formats when we can move on. Fortunately users are quite willing to move on with us.

The dependency on Lua is not so much a dependency as well as progress, especially if we keep in mind that the development of LuaTEX has been driven by the ConTEXt people. It's way more fun to use the TEX, Lua and METAPOST languages for what they are best suited for than to try to squeeze all functionality out the macro language only. It also suits the original design goals that Don Knuth gave TEX: to take the code and adapt it to ones needs.

An interesting side effect of the multilingual interface is that it made the low level key/value handling more efficient. Given that we nowadays have faster machines in MkIV some of the gain in speed has been sacrified to a more neat but slower inheritance subsystem. Probably no user will notice this anyway.

Being monolitic

When you run ConTEXt you don't need to load (or setup) extra styling code. You will always get some output. For those who come from other macro packages this is somewhat hard to grasp and sometimes seen as a disadvantage.

However, a fact is that in practice one can easily modulate on the defaults.

```
\setupbodyfont[dejavu]

\setuplayout[width=middle,height=middle]

\setupwhitespace[big]

\setuphead[chapter][style=\bfc]
\setuphead[section][style=\bfb]
```

Is the above really much more work than loading a style that defines the font and another one that sets up the spacing and styles? Also, a user can put these commands in a file and load that one. Changing the look and feel this is way more convenient than loading some default and try to overload unwanted settings (especially if that style changes). It also gives the user an idea that there can be a personal touch to the document. Of course the user can just stick to the defaults.

Any observation that users are supposed to know plain TEX or do some coding is just wrong and probably come from experiences with other macro packages. On the other hand it might help the user to know a bit about the project structure, separating structure and layout and limiting coding. Much in ConTEXt relates to structure and the actual rendering is an independent issue. Of course a user can still do things similar to plain TEX, so buying a copy of The TEX Book is no waste: you can use most tricks mentioned there in ConTEXt and there is a lot of information about fine-tuning math typesetting. It also does not hurt to know a bit about where we come from.

Speed

Indeed ConTEXt is not a fast runner, but it's not that slow either. In some cases a slow terminal is the culprit (as TEX does no buffering), and in other cases the user just asks for something that needs processing time. Especially decorating the page will increase the runtime. Of course delegating some action to Lua costs time, but we gain back functionality that otherwise would not be possible or take much more runtime. The startup time of MkIV is much shorter than MkII which is partly due to more efficient file searching so in practice MkIV runtime is quite acceptable, espically if we consider that we load larger fonts and operate in a Unicode universum. Also, hyphenation patterns are loaded only when needed and, when used, METAPOST processing happens instantaneously.

Development

Indeed most development is done by a few people, but how bad is that? If we look at the larger picture, there is a whole infrastructure in place: wiki, stand-alone distribution, mailing lists, conferences, and all hat is taken care of by pretty active crowd. The advantage of a small development team is that consistency is easier to guard and (hopefully) the distance between developers and regular users is rather small. Also, input with regards to subsystems (language labels, font definitions, etc.) is accepted from whoever comes up with improvements, given that they are consistent. There are no formal committees and reasonable demands are often met within reasonable time.

Specific fields

There is no reason why you shouldn't use ConTEXt if you need to typeset math. First of all it runs on top of TEX so you will get your math done one way or the other. Also, math support is quite complete although the implementation differs from other macro packages (this might be even more true in MkIV). We're not bound by traditions and have some pretty good experts on board that helps us to move on. And more is coming.

Another area of interest is critical editions. Again some experts in the field are involved and ConTEXt keeps evolving in this area, driven by demands from advanced users.

All the usual bells and whistles that TEX engines can offer, like character protrusion and hz optimization are supported as are advanced font features. Also, we already had quite some specialized functionality for detailed typesetting and much already has been rewritten in MkIV. Quite some languages are supported right out of the box. However, keep in mind that this is all built in so don't start looking for special styles or so (as the lack of them might suggest that it's not there). Just join the mailing list.

Kernel code

Users normally stick to the more high level commands. There are however quite some support macros. The real low level code is not be touched by users. Users coming from other packages might need to get accustomed to leaving them along and define (and tweak) instances at the higher level. In MkIV most of ConTEXt is also available at the Lua end and sometimes solving more complex problems is done easier that way. By now we have a hybrid system. As part of the rather drastic MkIV cleanup and rewrite we're in the process of hiding obscure code from the user and might end up with better low level api documentation.

On top of the kernel code we have some modules. These implement very specific functionality and more and more get written. There are for instance modules for typesetting MathML, loading fonts with less commands, integrating external applications, (advanced) presentations, tracing. They are good examples of how to write extensions.

Documentation

There is quite some documentation but it's rather diverse. The documents written by the authors are often a side effect of some development. There is a shift towards more general documentation (and even printed copies are available) but writing can get a lower priority in a time when quick and dirty answers can be found on the internet, mailing list or wiki. On the other hand, the user interface has always been pretty well defined in a formal way) and once a user knows what some keys do with a command, he or she can also use that knowledge for other commands. And, as we aim for upward compatibility, a decade old manual might still apply so there is no need to render it again just for the sake of updating a date.

For what it's worth: whenever I have to solve a problem with a program (or language implementation) I run into cases where I have to look long to find (non conflicting) information. It just comes with the problems one wants to solve and TEX (ConTEXt) is not different.

Conclusion

So let's draw some conclusions and limit ourselves to MkIV. First of all this is the versions that (new) users are supposed to use. It also means that they use LuaTEX, which in turn means that there are no further dependencies. The ConTEXt suite that you can download from `contextgarden.net` is an easy starting point and TEX Live is also an option. They should just start and delay styling to when they feel the need. By that time they only need a handful of commands to get a decent job done. If you can't find the documentation you need (on paper of on the wiki), consider participating in writing it.

Hans Hagen
PRAGMA ADE
Hasselt NL
October 30, 2011

www.pragma-ade.nl
www.contextgarden.net