Do we need a font system in TEX?

Hans Hagen
PRAGMA ADE
http://pragma-ade.com

Abstract

In this article I will reflect on the font system(s) currently built into ConTEXt. From this perspective I will mention the current directions in the development of TEX engines and how they may influence ConTEXt. I will also put this in the context of document layout design.

1 Introduction

This article was written while Taco Hoekwater and I were working on LuaTEX and ConTEXt MkIV, work that is ongoing. This process gives us much time and opportunity to explore new frontiers and reconsider existing ConTEXt features. We also use this process to write down some ideas as they evolve. Much detail is missing and I assume that the user is somewhat familiar with fonts.

Since this article is not typeset using my regular TEX setup I will not give many examples. After all, it's just meant as a teaser for users who want to discuss future font support in TEX (especially in ConTEXt). In the related presentation I will give a few examples.

2 Starting point

One of the characteristics of a TEX macro package is that it provides some kind of font system. Such a system deals with two issues:

- consistent switching between different styles and sizes (mostly in text mode)
- handling relative scales of fonts in super- and subscripts in math mode

The TEXbook and its related plain TEX format demonstrate what such a system may look like. However, it sets up a 10-point system and when users want, for instance, a 12-point setup, more definitions are needed. As soon as a user switches between 10 and 12 points a whole set of commands needs to be redefined or at least commands need to adapt their behaviour.

3 Into context

The macro package ConTEXt evolved over time and in principle permits you to set up your own font system, but in practice users will use the built-in font support which is organized as follows.

The main classification is *style*. Examples of font styles are serif (rm), sans (ss) and mono (tt) but math (mm) is also a style. Of course this is a rather arbitrary classification, but it's kind of rooted in the fact that Computer Modern came in these variants. When I started using the Lucida fonts I introduced handwriting (hw) and calligraphic (cg) styles and more are possible.

Next we have *style alternatives* such as normal, slanted, bold, italic and the like. Again, these are rooted in the fonts that came with TEX. Slanted is kind of artificial and italic is not always really italic, which is why the term 'oblique' is used as well.

Then comes *size*. In addition to the normal size one can switch between predefined but configurable additional sizes: larger ones denoted by the characters a, b, c, etc., and smaller sizes by x and xx.

Font switches are either written as part of the source stream or set as property of (structural) elements. Examples of stream commands are:

```
\rm \tt \tf \tfx \bf \bfx \bfxx
\sl \sla \slb \tttfx
```

Style properties are defined and used like this:

```
\setuphead[subsection][style=bold]
\definefontalternative[LargeAndBold][\bfd]
\setuphead[section][style=LargeAndBold]
```

If a user is in control of the style, such a system works rather well. One can conveniently switch to a different style, alternative and/or relative size.

4 Typefaces

When one is not in control of the document design, there's always a chance that one has to deal with yet another level of organization. Think of a journal where some articles are typeset with FontFont Meta for the running text combined with Lucida Math, and other articles are typeset in Palatino for both text and math. Add to that yet another choice of fonts for the headers and footers and we're talking of three distinctive font setups for one publication.

This is where typefaces come into play. These are combinations of styles within one collection. One can for instance define a typeface *palatino* which is a combination of Palatino Nova (serif), Palatino Sans

(sans) and Palladio Px (math) completed with Latin Modern Typewriter (mono). Of course we need to make sure that we scale the Latin Modern to match the Palatinos. The following definitions were used for the reader of the ConTEXt conference in Epen (2007):

```
\definetypeface[mainface] [rm][serif]
  [palatino-nova-regular] [default]
\definetypeface[mainface] [ss][sans]
  [palatino-sans-regular] [default]
\definetypeface[mainface] [tt][mono]
  [latin-modern-light]    [default]

\definetypeface[extraface][rm][serif]
  [palatino-nova-regular] [default]
\definetypeface[extraface][ss][sans]
  [palatino-sans-informal][default]
\definetypeface[extraface][tt][mono]
  [latin-modern-light]    [default]
```

These are applied with:

```
\setupbodyfont[mainface]
\setuplayout[style=
  {\switchtobodyfont[extraface,sans]}]
```

The `default` parameter selects the scaling model, in this case not based on design sizes, but derived from 10-point variants.

To make life (and choosing) even more complex, users more and more run into fonts that come in different weights (light, regular, medium, dark, ultra), thus ending up with multiple typeface definitions becomes the norm. It also means that users will always have to face the difficulties of font definitions: the burden of too much choice. What combination looks best?

```
\starttypescript [mono]
  [latin-modern-regular] [name]
    \usetypescript[mono][fallback]
    \definefontsynonym[Mono]
      [lmtypewriter10-regular]
    \definefontsynonym[MonoItalic]
      [lmtypewriter10-oblique]
    \definefontsynonym[MonoBold]
      [lmtypewriter10-dark]
    \definefontsynonym[MonoBoldItalic]
      [lmtypewriter10-darkoblique]
\stoptypescript

\starttypescript [mono]
  [latin-modern-light] [name]
    \usetypescript[mono][fallback]
    \definefontsynonym[Mono]
      [lmtypewriter10-light]
    \definefontsynonym[MonoItalic]
      [lmtypewriter10-lightoblique]
    \definefontsynonym[MonoBold]
      [lmtypewriter10-regular]
    \definefontsynonym[MonoBoldItalic]
      [lmtypewriter10-oblique]
\stoptypescript
```

Did I discuss design sizes yet? Computer Modern comes in design sizes. Apart from the esthetic aspect, this made much sense in a time where bitmap fonts were the rule. I must admit that I have no other fonts on my machine that come in design sizes. The core font system of ConTEXt is set up with design sizes in mind, but later extensions made defining typefaces based on one design size convenient (normally 10 point). For this reason users will never deal with the low level font definition system directly.

Recently we see design sizes come back in another disguise. Instead of variants in terms of size we get 'caption' and 'display'. Technically one can embed different design sizes in an OpenType font but this does not happen often yet.

5 Simple definitions

Occasionally we needed a special font definition, for instance when typesetting a title page. There we can use definitions like

```
\definefont [TitleFont] [SerifBold sa 3.5]
```

This means as much as: define a font *TitleFont* which uses the current SerifBold (symbolic names are used all over the place in the definitions, aka typescripts) and scale it to 3.5 times the current bodyfontsize. This means that we're freed of hard coded (and cryptic) font file names.

6 Features

One thing to keep in mind when setting up fonts is the font encoding. An encoding is a subset of glyphs out of the whole repertoire available in a font. Font encodings (not to be confused with file encodings or input regimes) are a side effect of TEX being an 8-bit system, a restriction which is removed by Omega (Aleph), XƎTEX and LuaTEX. Other characteristics are mappings (from upper- to lowercase and reverse) and, more recently, features as part of OpenType fonts.

For typesetting the mentioned reader I used LuaTEX in combination with the experimental ConTEXt version MkIV and so the OpenType variants could be used. The fact that the font itself provides features puts some demands on the font system. How do we pass them to TEX (in the case of XƎTEX) or Lua (in the case of LuaTEX)? In XƎTEX one can say:

```
\font\MyFont=
   palatinonova-regular:liga;dlig; at 12pt
```

But this does not go well with the abstraction and separation of name and style in ConTEXt. In LuaTEX one can implement any interface but at the price of taking care of translating features defined in the font into something that TEX can deal with. This is a fundamental difference with XƎTEX: it takes a macro package writer more effort to provide font support in LuaTEX, but this is compensated by more flexibility. As with XƎTEX we expect macro package writers to take care of that.

Recent versions of pdfTEX also introduced features, like *hz* optimization and protruding. These features can be applied to individual fonts as well as to styles and typefaces. For this we can use the 'font handling' subsystem that we will not discuss in this article. In short, that subsystem deals not with real font features, but with TEX applying its own features to a font.

In pdfTEX one can add inter-character spacing to a font using the low level commands:

```
\font\MyFont=somefont at 12pt
\letterspacefont
   \MyLetterSpacedFont=\MyFont 50
```

This means that 0.025 em is added on each side of a character. The problem with this TEX feature is that it refers to an already defined font. Also, one has to compensate for spacing before and after the sequence of characters manually. What complicates matters even more is that each feature uses a slightly different interface and that features are applied to global font definitions. Such low level commands are not something the average TEX user wants to deal with so we need some kind of high level interface.

Because the distinction between font features and TEX features is somewhat fuzzy, we will use the term features for both. From the user's perspective it does not really matter.

7 Interface

For a ConTEXt user, a more natural interface is the following:

```
\definefeature[myfeatures]
 [ligatures=yes,oldstyle=yes,spacing=.025em]
\definefont[MyFont]
  [somefont][feature=myfeatures]
```

How do we implement these and other features? Fonts can have small caps and oldstyle numerals. One may want these but not always. Here we face a dilemma: do we need a complete small caps typeface (many definitions) or is it just an alternative selection of glyphs. When we set up the base font system small caps were often of limited availability, so it ended up as an alternative by default. However, now that we have enough memory in our machines, and now that fonts often come with small caps in all styles and alternatives, we can equally well define it as an additional typeface. So, we can define a *palatino* alongside a *palatino-sc* and *palatino-os*.

Consider the regular shapes. In this case a macro package can decide to create three fonts out of, say, PalatinoNova-Regular: a normal one, one with lowercase characters replaced by small caps, and one with digits replaced by oldstyle numerals. But the package can also decide to pass the font as it is to TEX and at some point in the typesetting process swap lowercase characters by uppercase ones, and/or replace digits. This saves two font instances at the cost of some extra processing. Because the design of the document often includes a consistent choice for oldstyle numerals, it makes sense to create the extra font here, but in the case of small caps I'm not sure which alternative is better.

You may wonder what this has to do with interfacing so let's give another example. Sometimes a large chapter or section head looks better when a bit of inter-character spacing is applied. Do we create a spaced font for just a few occasions or do we move that to internal (node) processing? Defining a truckload of extra fonts just because we want to space a few times does not really make sense. Also, a spaced font is no real solution because one has to deal with the begin and end of a spaced sequence then.

What does a user actually want to tell the system? Is it:

```
some text {\UseMyLetterSpacedFontHere
  some text} some text
```

or maybe:

```
some text {\LetterSpacedThisText
  some text} some text
```

In the first case the user asks for a font switch, but in the second case we're dealing with a property which is not really related to a font at all, apart from the fact that the spacing may depend on font characteristics. I can also envision several variants: spacing based on character kerning, or equally spaced fonts, or slightly randomly spaced.

Or consider that at some point you want to use the outline variant of a font. This is a drawing property, not so much a font property, so again, the second approach may make more sense.

So, certain features may influence the interface as well: are we talking of a feature attached to a font definition, or is it applied to a range of characters (glyphs) in the document? In the first case we need to enable the feature when we define the font, but in the second case we can do that in the style when it's needed. What do users prefer most?

8 Frontends

For over 25 years the TEX engine was essentially frozen. With ε-TEX, some programming features were added, Omega added directional typesetting and pdfTEX built in the backend. Speaking for ConTEXt none of them really demanded a redesign of the macro package as a whole or one of its subsystems. Even XƎTEX with its font features could be supported rather easily until the moment that the name specification was extended to support font names as well as filenames at which point the low level interface (using brackets) started interfering badly with the ConTEXt user interface. The greatest differentiation was in the handling of backends and that was implemented by separating specific backend code into driver files (think of color support or graphic inclusion).

The differences in frontends were negligible and could be dealt with by code branches or selective macro definitions at format generation time. However with the evolution of font systems, the frontend part became more tricky, and not only from the perspective of user interfacing. Suddenly we were dealing with features being present or not, or being implemented differently. So, from now on, even with a consistent user interface the users need to be aware of what exactly is supported by the frontend and with the font itself. We have to see where this leads.

9 Math

We have hardly mentioned math, so how about it? A substantial part of TEX and therefore its font machinery is dealing with math. Math in TEX is a family business. A family groups fonts in sizes: normal, small and smaller. Following the Plain TEX tradition we use families for math roman, italic, symbols, extension symbols as well as what we previously called alternatives (bold and so on).

And there the problem strikes. First our population only counts 16 families, which is not enough to deal with regular, slanted, italic, bold, bold italic, all kinds of extra symbols, also in variants, and more. Another complication is that one may want to use bold text but not bold math or the reverse. Add to this that TEX is programmed in such a way that changing families mid-formula is not an option (the last definition counts), you can imagine that it's hard to please users in this area. More families would make life easier, but that only partially balances the equation of demand and supply. Font encodings also may play a role here: specific math encodings and regular text font encodings (not all math documents are written in English).

10 Daily practice

If after this exploration you're still with us, we're ready to review this system. Over the years the ConTEXt user base has widened and the range of applications is impressive. This also means that we need to provide the current font related subsystems in future versions. Where do we stand with a font system that is set up for consistency and convenient definition of fonts in terms of base characteristics?

My own application of ConTEXt ranges from special applications, via manuals, to (often) fully automated generation of documents as part of a bigger workflow. For the last group of applications we have to provide the mechanisms as well as the styles. Most of the styles that I have to define are prototyped in desktop publishing applications. Not only is any systematic approach to using fonts missing, also many fonts are mixed together. This means that in practice one can forget about a proper font system. Of course I try to fit these into some kind of system, but since the input is often rather simple too, font usage is also predictable. I frequently end up with a simple typeface definition for the main body font where I also define the math and monospaced variants, because one never knows what fallbacks are needed.

Life is actually worse: designs are seldom consistent in terms of font usage, color application, (interline) spacing, layout and structure. But these are the cornerstones of ConTEXt and that means that in such cases they are much of what ConTEXt provides (no big deal because we have hooks all over the place).

11 Open type

At the same time we see OpenType fonts showing up and these provide features that were not available and/or were distributed over multiple fonts. On the one hand, these (often Unicode) base fonts are great, especially when used with a modern TEX implementation. Quite often I get specifications in a way that indicates that the designer thinks in terms of her/his application. For instance, when 10 pt is specified, in most cases 10 bp (or PostScript points) are meant. And is an 'H-height' the same as an 'X-height'? I'm not sure that the abundance of features in OpenType fonts will be dealt with consistently and with care.

Here is an example: schoolbooks that teach kids French are typeset using systems that come preconfigured for English. Suddenly fonts have language-related features and you can bet that these are used. However, in the past, awareness of such features is dim. How many schoolbooks use the proper French spacing around colons and semi colons? And how many use the right French quotation symbols? If it does not happen today, how about the future? How consistent will designs be? Just watch how suddenly we see those relatively unknown ligatures (like st) show up, simply because they are there. The fact that these are language dependent does not bother some users.

As it happens, support for languages in TEX has always been quite strong. Users are aware of their language needs, and TEX supports them. Actually, in many areas TEX provides a lot of detailed control, and this may conflict with less sophisticated control driven by fonts. We cannot assume that all font designers and foundries pay an equal amount of care to each (often big) OpenType font.

The number of available math fonts is not large. This means that when those are converted to OpenType and use the Unicode encoding, we can get rid of many nasty tricks at the macro level. There will be no more need for tricky family magic, nor for font switches at unfortunate moments: we only have a few fonts left. Because LuaTEX provides a way to define virtual fonts on the fly, missing bits and pieces can be filled in and style alternatives can be provided even if the math fonts themselves lack them. Of course this only works out well if we are willing to rewrite and/or extend parts of macro packages substantially.

12 Control

So, in addition to the question whether we need a full-blown font system, we need to ask ourselves where we let the font drive the machinery (the font controls TEX) and where we let TEX be in control (TEX controls the font). In LuaTEX we (the LuaTEX team) provide access to the font definition mechanisms, which permits macro package writers to let the font be the driving force. For instance, one can define a font complete with ligature information and let TEX do the job. But one can equally well bypass this mechanism and process node lists (one of TEX's internal representations of the typeset text) by using special Lua code hooked into TEX. Or take the mentioned kerning around French punctuation: this can be a font property but also a matter of node processing. Because most TEX users leave such details to macro packages, one can expect both solutions to show up. Instead of hard coding alternatives in the TEX kernel, we just provide the machinery to macro writers.

Recently I had to write a style for a project and rewrite it many times because automated typesetting suddenly forces those involved to pin down designs. It's often hardly a challenge for a TEX user to identify the inconsistencies between different volumes of a series of books (equally well one can identify systematic problems with TEX macros because they show up each time). When reverse engineering an existing design inconsistencies creep in, and quality control depends on which volume is taken for comparison today. In this case it also happened that the design of this series was based on a font that was not only very incomplete, but also buggy. Familiar characters were missing, names in the encoding vector were wrongly applied. So, we had to come up with a special font encoding that in itself was wrong with regards to the names used. This is a bit of a nightmare because a different encoding results in extra map files as well an extra instance of hyphenation patterns, i.e. another format file.

In LuaTEX this can be done differently. There one can add some code to the loader that takes care of special remapping and filling in gaps with placeholders. Of course this can be embedded in a higher level user interface. I already have quite a lot of experimental code marked to be turned into production code some day.

13 Conclusion

When dealing with complex and/or very structured documents we can benefit from a font system as currently found in ConTEXt. Users can be sure that when they switch to another style or alternative, that the system will follow.

But what about a font system for situations where TEX has to compete with (or replace) desktop publishing? There we can roughly conclude the following.

- We can stick to a simple font model: one size for the main text, a few definitions for different alternatives (regular, bold, italic, bolditalic) because this is what the designer has available.
- In addition we have to define a truckload of fonts for all kind of elements (structure, ornaments, bits and pieces of the page body, captions, tables, etc.). We can stick to dumb font switches since the (structural) editing tools used don't permit anything beyond the specs anyway.
- Small caps, oldstyle numerals, inter-character spacing in titling, and so on can be applied

when constructing an internal font representation or handling can be delayed to node processing time. Depending on the quality of the font, some tweaking needs to be done. It is still open whether we treat them as font features or as a property of a part of the text.

- Math is a different story. If dealing with third party input, it's often more a matter of cleaning up than of advanced font trickery. Unicode math fonts may simplify our life but may as well complicate it. But whatever solution we end up with, more families are welcome. We need to be prepared for exceptions (especially when dealing with specialized math, schoolbook math) and also need to keep in mind that TEX no longer dominates this market or at least is fed with input coming from word processors with math editing capabilities.
- Advanced features like hz and protruding are of course possible but will often be interpreted as errors by QA people. Applying them in TEX is not complex, but explaining them to designers may be. Anyway, in most cases ragged right is to be used, if only because designers don't trust systems to do a proper justification. Here TEX's 25 year reputation of creating nice paragraphs does not help much.

It goes without saying that a simple font system will be faster than an advanced one normally used in TEX. So, any time that we lose in processing node lists, we may well gain back in a simplified font system. On the other hand, life may become more complex now that TEX engines provide more (distinctive) font related features, which in turn may drive user demand into all directions possible: *I want these ligatures but not those!* This may be compensated for by the fact that we need to load fewer fonts, and get rid of font encodings and character/glyph fall-back trickery.

In retrospect, the way the plain TEX format defines fonts is not that bad for most situations where some third party is responsible for the overall document design. The complication arises when one writes manuals and needs to switch frequently between sizes and styles.

Actually many dirty tricks used in macro packages also result from the simple fact that one needs to typeset user manuals about TEX, which means that one has to deal with characters in special ways which in turn may be reflected on the font system.

Of one thing there can be no doubt: the landscape of font usage is changing and TEX macro packages have to adapt.