Font Variants
A new ConTEXt feature for organising rich fonts

Abstract
Font Variants are a new (meta-)feature in ConTEXt, offering the opportunity of easier access to advanced or unusual font features, without the hassle of using full typescript switches. This article briefly runs through the basic theory and practice of so–called font variants, and gives a few strategies for adapting it for your own uses.

What are font variants?

Font Variants are a somewhat nebulous concept, hard to define precisely. They can be any 'variation' on a main font, where the variation can be glyph shape, weight, width, or other font feature. Often, variants share the same *characters*, but have different *glyphs*, and therefore different encodings. Other times variants will be a visual variation sharing the same encoding.

This distinction between features and encodings can be blurred when you consider different fonts may have different strategies for encodings. Consider the different strategies needed with a source font that contains old–style figures and small caps glyphs within the same font as lining figures and normal lower case, and one that has those characters within a 'small caps' font. This is one of the reasons why font variants were introduced, to provide a layer of abstraction to hide the font–specific details on how the variants are implemented.

At surface level, a font variant is a temporary font switch, to be used as a grouped command, with a user–determined argument determining the variation to select. Font variants may exist for each font style (e.g., `\rm`, `\ss`) and alternative (e.g., `\bf`, `\it`), but the features do not 'stack' atop one another. A bit of code for illustration, with the results in figure 1:

```
Hello all. This is a bit of text written for
MAPS \#32 on {\Var[osf]31} March
{\Var[osf]2005} in {\it Lancaster,
\Var[sc]uk}. {\bf I can’t \Var[lt]expect}
accumulative {\Var[lt]effects} with
variants like{\Var[cond] condensed
{\Var[sc]caps}}, however.
```

Hello all. This is a bit of text written for MAPS #32 on 31 March 2005 in *Lancaster,* *UK*. **I can't expect** accumulative effects with variants like condensed CAPS, however.

Figure 1. Antykwa Toruńska, with variants

The command `\Var[]` is shorthand for `\variant`, which can be used in case of a name clash (e.g., from mathematics).

Example usage

A couple font variants have already been named within current ConTEXt distributions, as the features were available in relatively new publically available fonts. Antykwa Toruńska is now a fully–featured serif family with many features available as variants. We see three classes of variation, in fact:

Glyph variation Beside each of the normal, roman fonts with lining figures, there lies a variant with variant glyphs: lower case letters are replaced with small caps, and lining numerals are replaced with old-style (lower case) numbers.

Weight variation Antykwa Toruńska offers four sets of weights: bold, medium, regular, and light. ConTEXt ordinarily allows easy access to only a normal and bold pair of weights, so the other pair can be accessed as a lighter (or darker) variant of the pair of fonts.

Width variation Antykwa Toruńska also offers condensed versions of the same fonts. These are defined as a typescript family themselves, but are also accessible as a condensed variant on the normal width. Conversely, the normal width is accessible from the condensed family as an expanded variant.

Usage then follows with the defined variants for the given family. For the normal width, normal weight Antykwa Toruńska family that is set up as follows:

```
\definetypeface[antt][rm][serif]
 [antykwa-torunska][default][encoding=texnansi]
```

...the following predefined variants are defined: 'osf' (old–style figures), 'sc' (small caps), 'lt' (light), and 'cond' (condensed). Markup proceeds with the `\Var[]` commands acting as grouped font switches.

Implementing variants yourself

There is one key command to set up font variants for implementors:
`\definefontvariant[style][name][suffix]`.
The *style* argument registers the variants with the Serif or Sans family of fonts. The *name* argument registers the name to be called within `\Var[name]`. The suffix argument reaches down within the typescripts and looks for a fontsynonym `style`*`variation`*`suffix`. For example, given a definition:

```
\definefontvariant[Serif][osf][-OldStyle]
```

and in an italic context, invoking the variant switch `\Var[osf]` means that that ConTEXt then tries to switch to the font `SerifItalic-Oldstyle`.

That simple name resolution means that the other half of implementing font variants consists of naming font synonyms within typescripts correctly. The Antykwa Toruńska typescripts in `type-syn` provide an extended example. There are a few simple guidelines to follow, however:

- You should always create a font synonym for `SerifRegular` to `Serif` and `SansRegular` to `Sans`. These are the fallback cases, and need to be accommodated by the somewhat simple–minded name resolution.
- After the fallback case, you need to define seven font synonyms per variant: Regular, RegularItalic, RegularSlanted, the same for Bold, and Caps.
- The synonyms can be arbitrary, so long as they resolve to actual fonts on your system. This is typically achieved via the encoding synonyms.

Design strategies

[N.B. This section is for people comfortable working with font encoding files, and understanding the occasional need to make one up. Don't be discouraged or distracted if you're not one of them.]

If you need to install a font yourself, then you may need to concern yourself with devising a new encoding, especially if you're dealing with a font that includes several glyph variations within the same font (e.g., *a, Asmall, a.swash*). You should go ahead and write your own encoding based on an existing one, for example a `texnansi` variant that has small caps and old-style figures. The name should then be `baseencoding-`*`variantname`*`.enc`, for ease of installation.

TEXFont has an option (*–variant=*) for using variant *encodings*, which causes it to look for an `.enc` file as above. This variant encoding is used in the creation of the `.tfm` file and within the `.map` file, but on the ConTEXt side, the font acts as if it were in the base encoding. This way, ConTEXt is able to use its internal encodings correctly, but externally (in the driver that deals with fonts), the actual glyphs are those selected in your custom `.enc` file.

Conclusion

Font variants are actually a fairly simple mechanism, but coupled with correctly–written typescripts, and by following some simple conventions, you can unlock the potential of today's increasingly sophisticated and rich fonts. Antykwa Toruńska has font variant support built into ConTEXt already, so you can start experimenting with support on a user level today.

Adam T. Lindsay
Lancaster University
UK
`atl@alum.mit.edu`