OpenType in ConTEXt

Multi–dimensional font support and advanced font features

Keywords
ConTeXt fonts OpenType

Abstract
This is a summary of issues encountered and solutions implemented in order to support some advanced OpenType features in ConTEXt. This article describes an accompanying set of support files that address installation (using TEXFONT), accommodating extended opticals families, and some "pro" font features. The extended character set afforded by pro fonts enables support for comprehensive small caps and old–style figures. Although the typescripts and commands are described together, certain features (like variant encodings for TEXFONTand optical typescripts) can be used independently of the other features described.

Introduction

This article, originally produced as a ConTEXt "My Way" article, introduces some issues in installing, using, and integrating OpenType fonts in ConTEXt. OpenType has been discussed elsewhere in detail, but it should suffice to say that it is a modern melding of POSTSCRIPT and TrueType, enabling advanced typography features and Unicode support.

RomanCaps and
ItalicCaps and
BoldCaps and
BoldItalicCaps
Figure 1 Small caps in several variants

With some of the most complex OpenType fonts, one also sees integration with multiple design sizes. This is not necessarily unique to OpenType fonts, but is a co–occurring feature seen with "premium"–quality fonts. Some of the advanced typographical features seen in such fonts are integrated glyphs for small caps (across every font variant), old style and tabulation figures, and greek glyphs, all within the same font. This article introduces some strategies for taking advantage of small caps and old style figures, and for rudimentary math fonts for fonts that include greek language support.

It should be noted that this package owes its existence to Adobe's "Type Classics for Learning,"[1] an education–only package of very high quality OpenType fonts, for 99 USD. Many thanks to Bruce D'Arcus for pointing out the existence of this package, and his enthusiasm and cunning for encouraging me to do something with it. He also pointed me to the work of Achim Blumensath, whose efforts in the LATEX domain sparked my initial ideas about font installation. Otared Kavian provided an extended mathematics example used in earlier drafts. George Williams and Eddie Kohler provided the actual font wizardry. Obvious thanks to Hans Hagen and Don Knuth for providing such a rich foundation for this modest addition to the TEX and ConTEXt canon.

The working assumption throughout this article is that users use ConTEXt with PDFTEX, which can handle the converted forms of each of these fonts. The support files (typescripts and sample encodings) are available at the author's web site.[2]

Font installation using FontForge

The first issue to deal with is getting the fonts to be installed into the TeX tree. Early on, I decided that I should try to use TEXFONT, because it was clearly a ConTEXt–friendly tool, it had excellent globbing support (e.g., for converting and installing an entire directory of fonts at once), and, frankly, because I was slightly more familiar with it than other tools. It was pointed out that FontForge[3], by George Williams, was a powerful, freely–available, open–source, cross–platform tool that handled OpenType fonts as well as any other. This first font installation method therefore depends on a working installation of FontForge.
I extended TEXFONT to include an optional pre-processing step that converts an OpenType font to a `.pfb` and `.afm` pair. From there, TEXFONT could work its usual magic.

The first option of interest to someone installing OpenType fonts is `-preproc`. This command–line switch triggers a run of FontForge for each `otf` found in the current directory.

For example, if you wanted to install the Cronos Pro OpenType fonts, go into the directory containing the fonts, and issue the command:

```
texfont --makepath --install --enco=texnansi --preproc --vend=adobe --coll=CronosPro
```

If you prefer to work with batch files in TEXFONT, the following line should be a good place to start:

```
--en=? --ve=adobe --co=CronosPro --so=auto --pre
```

In order to support at least some of the many extended characters in a “Pro” font, another command–line option was added to TEXFONT, `-variant=blah` (abbreviated as `-va=`), which allows one `.enc` file to masquerade as another. For example, if I am working with the `texnansi` encoding, I can create a *variant* encoding that substitutes small caps for the lowercase letters. I name that encoding `texnansiSC.enc`, put it in a place where it can be found (like fonts/enc/dvips/context under TDS 1.1), and run:

```
texfont --en=texnansi --va=SC --pre --ve=adobe --co=CronosPro
```

The `-variant` option appends the variant’s name (`SC`, here) to the end of the name of the encoding (`texnansi`), and looks for that particular `.enc` file on the path. One variant, `texnansiOSFSC.enc`, is included in the support files as a starter. It was the only variant I personally needed for initial support of my own fonts, but you’re welcome to create (and share!) your own.

It is worth noting that there is nothing about the `-variant` option that is intrinsic to OpenType fonts. If you are working with any sort of font with extended glyphs (such as Swash Caps), you can create a font that accesses those extended glyphs (while masquerading as a known encoding) by using this method.

Font installation using LCDF Typetools

For those who don’t have the patience to hand–assemble custom encoding variants, a more palatable option will be to use the LCDF Typetools[4] from Eddie Kohler. These tools have advanced, specialised features with respect to OpenType. The `otftotfm` tool is very capable of installing fonts by itself, but integration with TEXFONT gives further automation (globbing again) and better integration with ConTEXt.
In order to use the LCDF Typetools within TEXFONT, you should add the command–line option `-lcdf`. This option alone will install the raw `.otf` files, which PDFTEX, as of this writing, can use and embed into PDF files, but cannot subset. A more likely option for most users will be to combine `-lcdf` with `-preproc`, which converts the `.otf` files into normal Type1 `.pfb` files, a more commonly usable format for modern TEX systems.

Eddie's tools' real strength lies in the ability to activate OpenType features and use all the glyphs in the font to their full potential. These features include non–standard ligatures, contextual swashes, small caps, number cases and spacing, and historical alternates. The tools accomplish this by creating a custom encoding and by embedding ligature information in the `.tfm` files it creates.

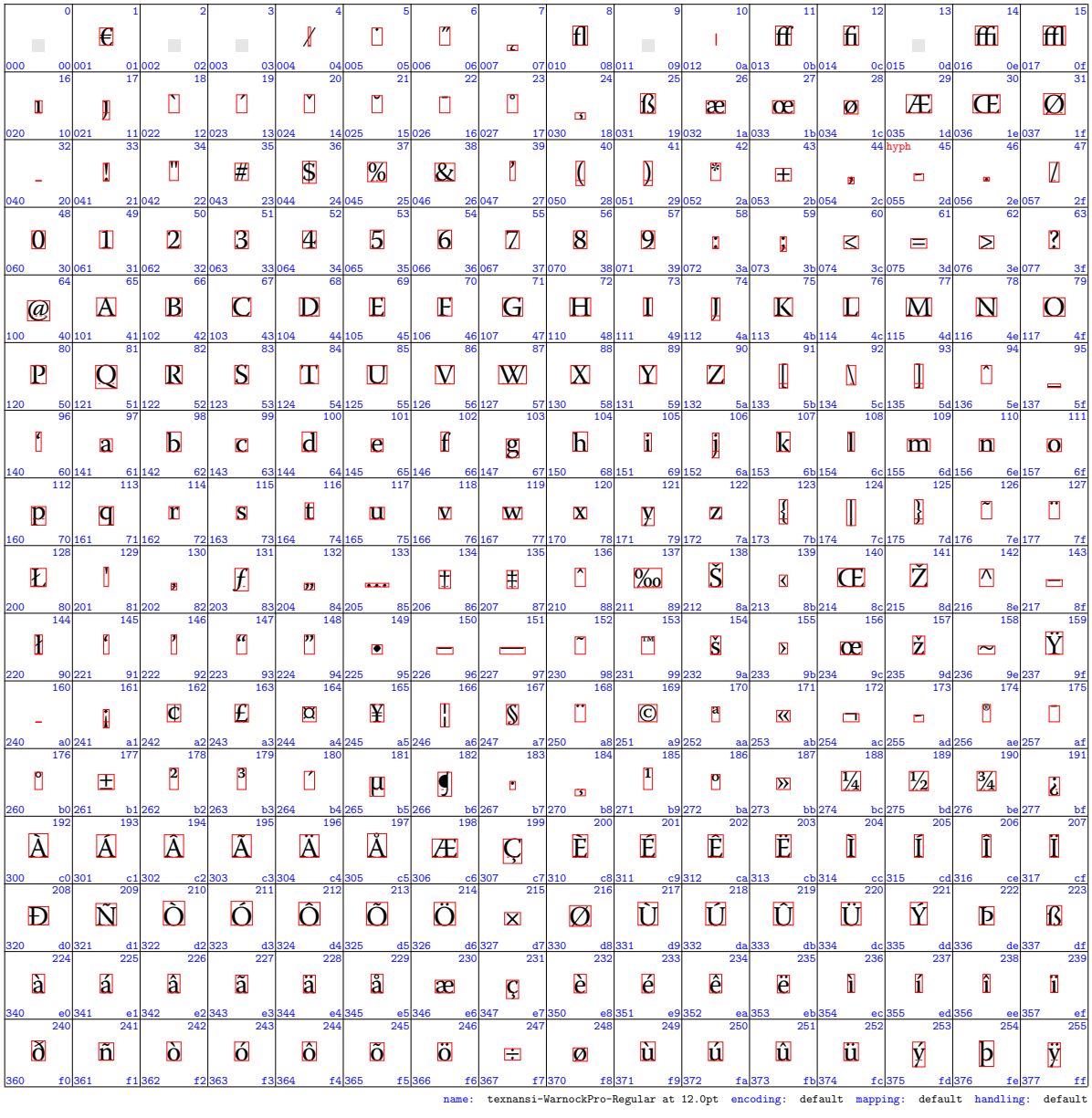

Figure 2 WarnockPro-Regular in the normal tex'n'ansi encoding

A TEXFONT user needs to be aware of two main things when using TEXFONT with LCDF Typetools. The `-variant` command–line option is re-interpreted as a means to pass four–letter OpenType features. For example, the command–line option of `-va=liga,kern,onum,pnum` passes the options of *ligatures, kerning, old–style numerals,* and *proportionally–spaced numerals* to the LCDF Typetools. The resultant `.tfm` files will be named: baseencoding-LIGA-KERN-ONUM-PNUM-fontname.tfm,

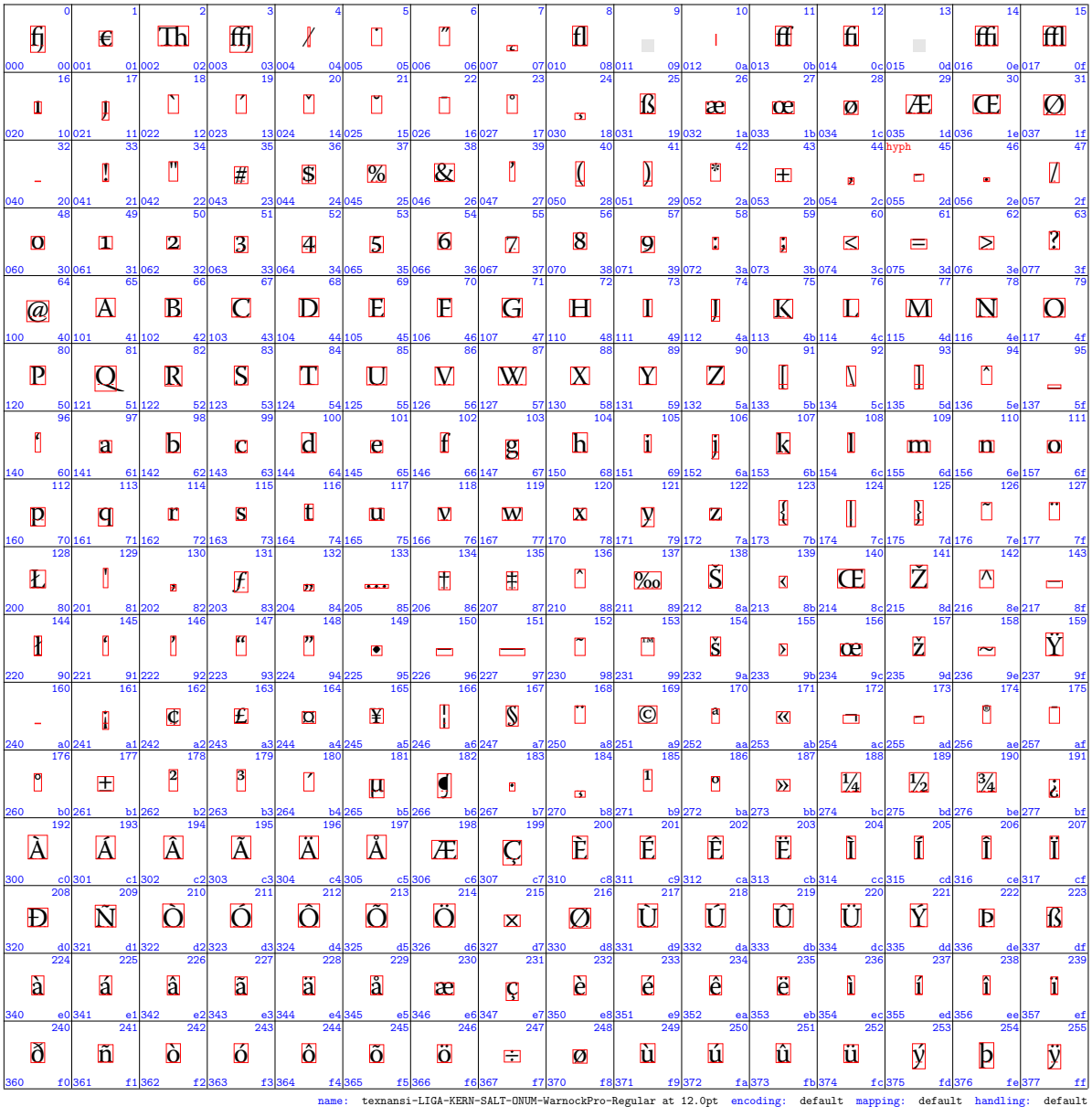

Figure 3 WarnockPro-Regular in the tex'n'ansi encoding, modified with the LIGAture, KERNing, Stylistic ALTernates, and Oldstyle NUMerals features. Note that not only are ligatures inserted into blank slots in the encoding, but the numerals and some letters (like the 'k', 'v', and 'y') are substituted with alternate forms.

and the produced `.map` file will be named: baseencoding-LIGA-KERN-ONUM-PNUM-vendor-collection.map.

The second thing to be aware of is TEX's limit of 256 glyphs in any font encoding. Many features (including ligatures and especially contextual swashes) require open slots in the base encoding. Although LCDF Typetools is clever in deciding which glyphs should be discarded from an over–full encoding, a user should keep this limitation in mind.

Opticals

Most fonts that TEX users are familiar with include only two design axes, namely weight (e.g., light, regular, or bold) and shape (e.g., italic, slanted, or roman)[5]. Back in the days of metal type, each size of a given font had different design characteristics, because the eye is sensitive to different features at different scales. "Optical" fonts essentially add another design axis. This design axis was well–developed in the days of multiple master fonts, but since that technology appears to be dying, premium fonts are now being issued at discrete points along that axis. The font package that was used in the development of these macros and typescripts typically included four optical font sizes for each font: caption, regular, sub–head, and display. Their differences are shown in figure 4.

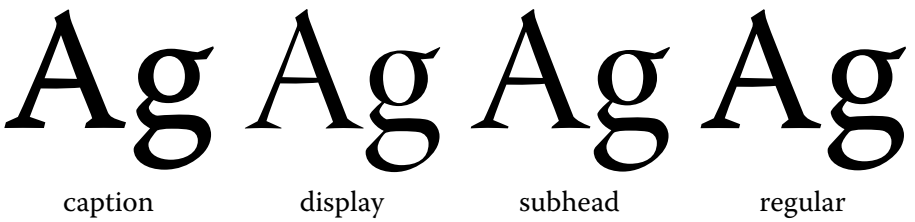

Figure 4 The four design sizes of Warnock Pro Opticals, shown at the same point size

At small design sizes (caption), there is typically lower contrast, a heavier stroke, a slightly larger x-height, and generally courser features. At large design sizes (display), strokes, tapers, serifs, and other details are much more refined.

This package supports these various design sizes with a series of extensive typescripts. It's essentially a brute–force method that defines font synonyms for each design size for each variant. That is, support for opticals is achieved by using a typescript that has small, regular, large, and extra–large fonts named for each of roman, italic, bold, and bold italic font variants, and adapting the `\tfa`–`\tfd` switching for each body font size.

For example, there are font synonyms declared for each of the following: SerifCaption, Serif, SerifSubhead, SerifDisplay, SerifItalicCaption, SerifItalic, SerifItalicSubhead, SerifItalicDisplay, and so on.... Each of these symbolic names is tied to font commands through the definition of a very large "Opticals" typescript, which generically associates a type variation and a type size with a design size. An extract follows:

```
\starttypescript [serif] [Opticals] [size]
\definebodyfont
  [12pt,11pt][rm]
  [tf=Serif sa 1,
  tfa=SerifSubhead sa \magfactor1,
  tfb=SerifSubhead sa \magfactor2,
  tfc=SerifSubhead sa \magfactor3,
  tfd=SerifDisplay sa \magfactor4,
   it=SerifItalic sa 1,
  ita=SerifItalicSubhead sa \magfactor1,
  itb=SerifItalicSubhead sa \magfactor2,
  itc=SerifItalicSubhead sa \magfactor3,
  itd=SerifItalicDisplay sa \magfactor4,
  ...]
\stoptypescript
```

As the bodyfont size changes (`12pt,11pt`, above), the relationships between the opticals change. This is handled in a huge typescript. In order to use this typescript yourself, you should define each of the SerifCaption . . . SerifItalicDisplay synonyms in your own typescript, and include that with the Opticals typescript.

For example, I defined the various opticals for the Warnock font in a typescript called `WarnockProSiz`. I first created a typescript that associated the optical sizes with the actual font names as installed by TEXFONT:

```
\starttypescript [serif] [WarnockProSiz] [texnansi]
 \definefontsynonym[SerifCaption][texnansi-WarnockPro-Capt]
                    [encoding=texnansi,handling=pure]
 \definefontsynonym[SerifText][texnansi-WarnockPro-Regular]
                    [encoding=texnansi,handling=pure]
 \definefontsynonym[SerifSubhead][texnansi-WarnockPro-Subh]
                    [encoding=texnansi,handling=pure]
 \definefontsynonym[SerifDisplay][texnansi-WarnockPro-Disp]
                    [encoding=texnansi,handling=pure]
 ...
\stoptypescript
```

I then created a Warnock typeface to tie my size synonyms with the Opticals typescript:

```
\starttypescript[Warn]
\definetypeface[war][rm][serif][WarnockProSiz][Opticals][encoding=texnansi]
\definetypeface[war][rc][romancaps][WarnockProSiz][Opticals][encoding=texnansi]
\stoptypescript
```

If SerifDisplay
you can SerifSubhead
read all of this SerifSubhead
without squinting at all, SerifSubhead
you have better eyesight than Serif
I do. In fact, these fonts can get quite Serif
small without losing all that much legibility! SerifCaption

Figure 5 The `\tfd...\tfxx` series with a base size of 12pt.

Small Caps

Current support for small caps, both inside and outside TEX, is generally very primitive. Most fonts – if they do offer it – only offer small caps support as a variation on the plain, roman font, and not for any italic/slanted fonts or bold fonts. This means that the small caps shape is of limited use with non–normal font alternatives (what ConTEXt calls `\it` and `\bf`). With a full complement of small caps shapes for each font alternative, small caps can be used more extensively.

The approach chosen for this package was to create another serif font style to exist inside the serif family, alongside the familiar roman (`rm`) style. The new font family, roman caps (`rc`) defines parallel font alternatives, using small caps variants. This means more work in terms of defining font synonyms, but it enables the small caps shape as a full design axis. The definitions begin as follows:

```
\definebodyfont
  [12pt,11pt][rc]
  [tf=SerifCaps sa 1,
  tfa=SerifCapsSubhead sa \magfactor1,
  ...]
```

. . . and proceeds as the other definitions, above. There are sans serif equivalents defined, as well. The sans small caps family (`cs`, caps sans) is treated the same way. If you have installed a small caps type variant for a sans serif font, you should define and use this parallel family.

THIS

Figure 6 `\rc\bf this`

In order to use these font families, you may call them directly with macros like `{\rc\bf this}`. This is inconvenient, and requires you to recall the font alternative as well as the roman caps font style. To alleviate the inconvenience, the support files for this article define a new font command, which switches from the normal style to the caps style while keeping the current alternative. The command is `\SmCap`, and it is used grouped, like other font commands. The following text in the margin is achieved with the code:

```
{\it text {\SmCap text \em text} text \fontstyle\ \fontalternative}
{\bf text {\SmCap text \em text} text \fontstyle\ \fontalternative}
{\rc text {\SmCap text \em text} text \fontstyle\ \fontalternative}
```

text TEXT TEXT *text rm it*
text TEXT *TEXT* text rm bf
TEXT TEXT *TEXT* TEXT RC TF

There is an identical command `\OldStyle`, which assumes that there are old style figures defined in the small caps family. It works in the same way as the above:

```
{\tf 0123456789 \OldStyle 0123456789}\crlf
{\bi 0123456789 \OldStyle 0123456789}\crlf
{\itx 0123456789 \OldStyle 0123456789}\crlf
```

0123456789 0123456789
0123456789 0123456789
0123456789 0123456789

Known issues

The sheer number of fonts defined means that these typescripts are extremely memory–intensive. A modern computer should not struggle, as long as the font memory allocated to ConTEXt is sufficient.

The default compound hyphen that is used in composed words is a hybrid character that doesn't work well in some pro fonts, including Warnock. I think it is best to replace the default compound character with a stretched hyphen with the command:

```
\def\compoundhyphen{\scale[sx=1.5]{-}
```

well–hung
hang–dog

Figure 7 ConTEXt's default compound hyphen vs. a stretched hyphen.

Math support is preliminary and still considered a work–in–progress, but it certainly seems possible to fashion a math font from sufficiently complete (e.g., containing Greek language support) fonts.

Footnotes

1. see http://www.adobe.com/education/ed_products/typeclassics.html
2. see http://homepage.mac.com/atl/tex/
3. née PfaEdit, see http://fontforge.sourceforge.net/
4. Available from http://www.lcdf.org/type/
5. Yes, it's true that the ubiquitous ComputerModern has many design sizes.

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