The PracTEX Journal
TPJ 2006 No 1
Rev. 2006–02–01

Mac OS X Fonts in pdfTEX

Gerben C. Th. Wierda · Thomas A. Schmitz · Adam T. Lindsay

Installing a new font with your TEX installation can be a challenging task. This article documents an attempt to provide an automated solution for users running Gerben Wierda's distribution of the TEX system on Apple's OS X. Using the fonts described here will only be possible for those who run this operating system; the way of making these fonts work with TEX should be of interest for all users. The article's level can be described as intermediate to advanced; it assumes some previous knowledge of TEX and fonts.

1 Introduction: It all began with a rant...

If you look into a TEX mailing-list or one of the newsgroups like `comp.text.tex`, you know the theme is a classic: using fonts in TEX. This time, it cropped up on the mailing-list "TeX on Mac OS X." In November 2005, some members complained they found it too hard to make the fonts that come with their system work with TEX.

At that point, Gerben Wierda privately contacted Thomas Schmitz. Gerben organizes the most popular (re-)distribution of the TEX-system for OS X. It is built on top of teTEX and TEXLive, but it also adds a couple of features that users have asked Gerben to add (such as additional fonts or the `prosper` class for LATEX). Gerben suggested that support for some of the fonts that come with the operating system should be made part of gwTEX. Soon, Adam Lindsay, who had already thought about a similar project, joined forces, and we spent a couple of days of frantic activity and hundreds of e-mail messages with assorted attachments until everything was in place and a first release of gtamacfonts could be made. This article will document what we did, how we did it, and how it was integrated into the distribution. We will give an outline of the philosophy behind it

all and of the general concept without going into the gory details here. Most of this technical stuff is amply documented elsewhere, e.g. in **Adam's article**[1] on OpenType fonts and **Thomas's article**[2] on TrueType fonts.

2 The General Concept

Support for the fonts involved two steps:

1. TEX needs a number of files in order to work with these fonts:

   - The metric files (`.tfm` and `.vf`) containing information about the boxes of the font's characters, the kerning, and the ligatures. All of this is needed by TEX to calculate where characters are to be placed;

   - a map file (`.map`) that declares the relation between these metric files and the font files that contain the actual outlines ("glyphs") of the characters (`.ttf`);

   - encoding files (`.enc`) that will allow us to extract a subset of 256 glyphs (shapes) from the fonts (which contain many more characters) per encoding (see below for an explanation);

   - for LATEX: font definitions (`.fd`) for different encodings and (optionally) packages (`.sty`) to facilitate choosing the fonts in your documents;

   - for ConTEXt: a typescript (`type-***.tex`) that organizes the fonts into families and allows using the normal font commands such as `{\em }` or `{\ss }`.

2. The fonts themselves have to be converted into a format that TEX can use. Macintosh systems typically put all the files for a font (regular, italic, bold, bold italic) into one big container ("font suitcase"). Mac OS X updated this concept with yet another type of container, the `.dfont` file. Ordinary TEX cannot work with these containers,[3] so

[1] `http://www.tug.org/pracjourn/2005-2/lindsay/lindsay.pdf`
[2] `http://www.tug.org/pracjourn/2005-2/schmitz/schmitz.pdf`
[3] But SIL International's XETEX can, and it also provides an interface to special aspects of the fonts.

the information needs to be extracted and put into regular font files. All the fonts shipped on newer Macintosh systems that are contained within `.dfont` files are TrueType fonts (`.ttf`) which can be used by pdfTEX, but not by `dvips`.

As we will see, most of the fonts demanded some special treatment, so we could not rely on a completely automated tool (such as `texfont` or `fontinst`) to do all the steps. We had to write quite a few of the files by hand, and we had to tweak some of the files that were produced by automatic tools.

3 Encodings

The first step was thinking about encoding vectors. An encoding vector defines a relation between TEX's internal representation of characters and glyphs (shapes of the 'characters' in a font). TEX is 8-bit-oriented and thus limited to encoding vectors with 256 positions. However, modern fonts may contain thousands of glyphs. A TEX encoding is therefore often a compromise on the availability of glyphs. Support for all the shapes in a font in TEX often means that it has to be broken up into many fonts, each with 256 possible glyphs. We decided we wanted support for the two most popular encodings:

1. EC, a.k.a. Cork encoding or T1 in the LATEX world. EC-encoded fonts offer numerous accented letters from many European languages. In LATEX, they are usually accompanied by a symbol set in the so-called TS1-encoding.

2. TEX 'n' ANSI, a.k.a. LY1 in the LATEX world. Texnansi has been more popular with ConTEXt users; it offers a reasonable number of accented letters and a selection of interesting symbols.

For most of the fonts, we could use the encoding files that come with teTEX. However, some of the fonts supply oldstyle numerals and/or small caps, so we had to define six additional encoding vectors:

☞ An EC- and texnansi-encoding that would produce oldstyle figures *and* small caps,

Compared to XETEX's interface, the method described here is less rich, but does not require a different TEX dialect, so it's more portable.

called `ec-sc.enc` and `texnansi-sc.enc`;

- an encoding that would produce oldstyle figures but regular lowercase letters (`ec-os.enc` and `texnansi-os.enc`);

- and, finally one that would produce lining figures and small caps, called `ec-sclf.enc` and `texnansi-sclf.enc`.

Unfortunately, this uniform approach did not work for the beautiful HoeflerText font (which has been used to typeset this document). Not only does the font have non-standard glyph names, but these names are not consistent across the regular, italic, and bold ("black") variants! We had to produce an entire set of EC- and texnansi-encodings for the variants of Hoefler.

4 Producing the Metric Files and the Fontmap

When we began producing the metric files from the fonts, we had to take one detail into account: TrueType fonts come in various forms. Macintosh TrueTypes have kerning- and ligature-information in parts of the files where the automated tools cannot find them.[4] That meant we could not use a tool such as `ttf2tfm` that would directly produce metric files from a TrueType file. Instead, we had to take a two-step approach:

1. Produce Adobe Font Metrics (`.afm`) from the fonts; these contain the kerning and ligature-information;

2. convert these `afm`s into TEX metrics, using one of the encoding files described above.

For step 1, we used the tool `fontforge`, written by **George Williams**[5]. It can produce a full `afm` from a Macintosh font, containing all the glyphs and the information about kerning and ligatures. Essentially, we used `fontforge` to convert the font from

[4] Before you leap to the conclusion that it's the case of the Macintosh platform being non-standard once again, consider for a moment that Apple invented the TrueType format.

[5] `http://fontforge.sourceforge.net/`

TrueType to PostScript Type 1. In the process, it produced an `afm`-file as well; we used this file and discarded the resulting `.pfb` file.

We could then take the resulting `afm`s and produce TEX Font Metrics (`.tfm`) and Virtual Fonts (`.vf`) with the tool `afm2tfm`, which is part of a regular TEX installation. You can feed this tool an encoding and an `afm`, and it will convert it to a pair of `vf` and `tfm`s. Since Adam and Thomas were both producing metrics, we had to be careful to be absolutely consistent with naming so that we could work in parallel. A run might look like this:

```
afm2tfm Didot.afm -T ec-sc.enc -v ec-sc-Didot ec-sc-raw-Didot
```

Where possible, `afm2pl` was substituted when generating texnansi-encoded files, as the tool avoided creating unnecessary virtual fonts.

There was one problem for which we could not find an automated solution: the generated HoeflerText metrics, for reasons we could not understand, never included ligatures for "fi" and "ffi." In the end, we edited the virtual fonts by hand and inserted two lines that define these ligatures.

After producing the metrics, we had to add a line to our mapfile (`gtamacfonts.map`) for every new `tfm`, in this form:

```
ec-sc-raw-Didot Didot 4 <Didot.ttf ec-sc.enc
```

After this was done, we now had a couple of dozen of new "fonts" (for TEX, every `tfm` constitutes a new font, even if it is just a variation of an existing font with a different encoding). In theory, you could now use them for TEXing with the primitive TEX `\font` command, but in order to use them with your favorite macro package, a few more files are necessary for everyday convenience.

5 Support for LATEX

In LATEX, support for a font usually consists of two types of files:

1. Font definitions (`.fd`) tell LATEX which `tfm`s constitute the same font family: which one is the roman serif, italic, bold, slanted, small caps font? Their basic structure is easy to understand, but LATEX needs one `fd` for every encoding. For,

say, the Baskerville font, we have thus three files: `t1gtamacbaskerville.fd` for EC-encoding, `ts1gtamacbaskerville.fd` for the accompanying symbol font, and `ly1gtamacbaskerville.fd` for the texnansi-encoding.

2. A LaTeX-package (`.sty`) usually contains just a few lines of code containing, e.g., a redefinition of the main font in the form `\renewcommand{\rmdefault}{gtamacbaskerville}`, which will make the font family defined in `t1gtamacbaskerville.fd` the roman default for the document. The packages distributed with our implementation contain a few extra features as will be seen below.

The font definitions had to be hand-written. But it's a task that can be done very fast: these files define every imaginable combination and weight, but our fonts typically have only four of them (regular, italic, bold, bold-italic, maybe small caps). So you need to fill in the names of these four `tfm`s; all the rest of the lines just tells LaTeX to make appropriate substitutions. As an example, here's the font definition for Didot in texnansi-encoding (we have not included all of the lines just defining substitutions):

```
\ProvidesFile{ly1gtamacdidot.fd}

\DeclareFontFamily{LY1}{gtamacdidot}{}

\DeclareFontShape{LY1}{gtamacdidot}{m}{n} {<-> texnansi-Didot}{}
\DeclareFontShape{LY1}{gtamacdidot}{m}{sc}{<-> texnansi-sc-Didot}{}
\DeclareFontShape{LY1}{gtamacdidot}{m}{sl}{<-> ssub * gtamacdidot/m/it}{}
\DeclareFontShape{LY1}{gtamacdidot}{m}{it}{<-> texnansi-DidotItalic}{}

\DeclareFontShape{LY1}{gtamacdidot}{cb}{n} {<-> texnansi-DidotBold}{}
\DeclareFontShape{LY1}{gtamacdidot}{cb}{sc}{<-> ssub * gtamacdidot/cb/n}{}
\DeclareFontShape{LY1}{gtamacdidot}{cb}{sl}{<-> ssub * gtamacdidot/cb/it}{}
\DeclareFontShape{LY1}{gtamacdidot}{cb}{it}{<-> ssub * gtamacdidot/m/it}{}

\endinput
```

As you can see, only four lines contain actual definitions: for regular (`m/n`), small caps

(m/sc), italic ("slanted," m/sl), and bold (cb/n). All the other lines define "silent substitutions" (whence the ssub *).[6] Didot doesn't have bold italic, so gtamacdidot/cb/it is substituted by regular italic, gtamacdidot/m/it.

Once you have such a template, preparing fds for other encodings and other fonts is just a matter of find/replace.

However, we wanted to give users a bit more help, so we added some more features to our packages:

- Many of the fonts we make accessible are sans-serif fonts. They will almost always be used to go with a regular serif-font. Since design-sizes of fonts differ a lot, we wanted to add the possibility for users to scale these fonts in order to obtain a homogeneous look in their documents. We copied the code for doing this from other packages, such as helvet.sty. Users can now call the package with a "scale" option:

  ```
  \usepackage[scale=0.92]{gtamacgillsans}
  ```

  Because of the way LaTeX sets up fonts, the code for using this feature must be both in the fd and in the package itself, which is why we have not implemented scaling for fonts that are already supported and have their own font definitions. This was the case for the Latin Modern font (lmtt) which we defined as the default mono font.[7]

- The ConTeXt typescripts define collections of fonts consisting of serif, sans, and mono typefaces. We did the same for LaTeX, so the main serif packages (baskerville, didot, georgia, and hoefler) redefine rmdefault, sfdefault and ttdefault. We also added a default scale to make the sans-fonts look good with the main serif font. For mono, we use Latin Modern.

- HoeflerText and Didot offer oldstyle figures. Many users prefer to have these figures made the default in their documents (such as in this one). For them, we have added an option to these two packages, for example:
  ```
  \usepackage[osf]{gtamacdidot}
  ```

[6] If you want to know more about the structure and syntax of fd files, have a look at section 7.10.3 of the LaTeX-Companion.

[7] In ConTeXt, scaling is implemented on the level of the individual typeface-definition; in LaTeX, it has to be present in the fd file as well. For the moment, we have decided to stick with the fd-files distributed with tetex, so we haven't implemented scaling for Latin Modern, but we may change our minds for the next release...

When this package option is used, oldstyle figures are default and lining figures are made available in the small caps font, accessible with the `\textsc` command.

There are now four LaTeX packages that define a serif, sans, and typewriter font:

```
gtamacbaskerville.sty
gtamacdidot.sty
gtamacgeorgia.sty
gtamachoefler.sty
```

Seven packages define a sans font only:

```
gtamacfutura.sty
gtamacfuturacondensed.sty
gtamacgillsans.sty
gtamachelveticaneue.sty
gtamaclucidagrande.sty
gtamaclucidagrande.sty
gtamacverdana.sty
```

And, finally, one package defines a typewriter font:

```
gtamacamericantypewriter.sty
```

6 Support for ConTeXt

ConTeXt support was a bit more flexible than that with LaTeX because there is generally less pressure to present all features with a unified interface. Every font's set of features is different, so why not give users those features as needed?

ConTeXt font support is generally achieved with typescript files (`type-*.tex`). These files include typescript definitions, which (for our purposes) fall into one of the following four categories:

1. Class–to–symbolic names that link a generic name (such as 'Serif' or 'SansItalic') with a name specific to the font family (such as 'HoeflerText-Regular' or 'GillSans-

LightItalic').

2. Symbolic–to–font names that link the specific font name above with an encoding–specific name that corresponds with an actual `.tfm` file on the user's system.

3. Map typescripts that trigger the loading of font map files. Map files create a relationship between the `.tfm` file that TEX knows about and a specific TrueType or Type 1 file. They are important as they tell the PDF building step where to find the font file that contains the glyphs.

4. Typeface typescripts group typescripts into family definitions. I would personally discourage using such preset definitions, as they have their limitations, but users find their convenience very compelling.

In the ConTEXt distribution, these groups of definitions are stored in different files (`type-syn`, `-enc`, `-map`, and `-exa`, respectively. In the gtamacfonts distribution, they are contained in one file (`type-gtamacfonts.tex`), for convenience.

The typescripts were written by hand, as each font family provided just enough variation to keep things interesting (and non–automatable). The real trick in typescripts is in the naming, keeping the typescript namespaces distinct, but overlapping enough to utilize the heavy redundancy across scripts. As an example, here are the typescripts for Gill Sans:

```
\starttypescript [sans] [gillsans] [name]
 \setups            [font:fallback:sans]
 \definefontsynonym [Sans]           [GillSans]
 \definefontsynonym [SansItalic]     [GillSans-Italic]
 \definefontsynonym [SansBold]       [GillSans-Bold]
 \definefontsynonym [SansBoldItalic] [GillSans-BoldItalic]
\stoptypescript
```

The above is a class–to–symbolic typescript: it points from the generic 'Sans' names to symbolic names specific to the Gill Sans font family. The `\setups` line is a compact, pre-defined way of making sure all of the alternatives within a font family (Slanted, BoldItalic, SmallCaps) point to default fallbacks if they are not over-ridden with an explicit definition we provide. For example, even though there is no slanted alternative listed, an `\sl` command would result in the SansItalic alternative, which, in the case

above, would resolve to whatever the GillSans-Italic font points to.

```
\starttypescript [sans] [gillsans-light] [name]
 \setups            [font:fallback:sans]
 \definefontsynonym [Sans]           [GillSans-Light]
 \definefontsynonym [SansItalic]     [GillSans-LightItalic]
 \definefontsynonym [SansBold]       [GillSans]
 \definefontsynonym [SansBoldItalic] [GillSans-Italic]
\stoptypescript
```

The above is a similar typescript that is triggered with a slightly different command. Instead of the normal alternative pointing to Gill Sans at a regular weight, it points to the light weight. The bold alternatives are similarly lightened to the regular variant.

```
\starttypescript [sans] [gillsans,gillsans-light] [texnansi,ec]
 \definefontsynonym [GillSans-Light]       [\typescriptthree-GillSansLight]
                                            [encoding=\typescriptthree]
 \definefontsynonym [GillSans-LightItalic] [\typescriptthree-GillSansLightItalic]
                                            [encoding=\typescriptthree]
 \definefontsynonym [GillSans]             [\typescriptthree-GillSans]
                                            [encoding=\typescriptthree]
 \definefontsynonym [GillSans-Italic]      [\typescriptthree-GillSansItalic]
                                            [encoding=\typescriptthree]
 \definefontsynonym [GillSans-Bold]        [\typescriptthree-GillSansBold]
                                            [encoding=\typescriptthree]
 \definefontsynonym [GillSans-BoldItalic]  [\typescriptthree-GillSansBoldItalic]
                                            [encoding=\typescriptthree]
\stoptypescript
```

The above maps the symbolic names to the low-level font file names. Because of the multiple values in the `\starttypescript` lines, it actually serves the role of four typescripts at once: two weight variations multiplied by two encoding vectors. If selected with the EC encoding, the first line would resolve to:

```
 \definefontsynonym [GillSans-Light] [ec-GillSansLight] [encoding=ec]
```

Since we followed a strict convention in naming the font files, we're sure that the

system will locate an `ec-GillSansLight.tfm` font file.

```
\starttypescript [map] [all] [all]
 \loadmapfile [gtamacfonts.map]
\stoptypescript
```

The use of the map typescript in the `type-gtamacfonts` package is very straightforward. If you load the typescript file, you end up triggering a catch-all condition, and the `gtamacfonts.map` file is automatically loaded.

```
\starttypescript [HoeflerOldStyle] [texnansi,ec]
 \definetypeface [HoeflerOldStyle] [rm] [serif] [hoefleroldstyle]
                                 [default] [encoding=\typescripttwo]
 \definetypeface [HoeflerOldStyle] [ss] [sans]  [gillsans]
                                 [default] [encoding=\typescripttwo,rscale=0.96]
 \definetypeface [HoeflerOldStyle] [mm] [math]  [palatino]
                                 [default] [rscale=0.90]
 \definetypeface [HoeflerOldStyle] [tt] [mono]  [modern]
                                 [default] [encoding=\typescripttwo]
\stoptypescript
```

The final typescript that uses the Gill Sans font is one of the example typescripts. Again, we would rather discourage their use, because the font combination was rather restricted (by those fonts we could be reasonably sure are present on a modern gwTEX/ConTEXt system) and idiosyncratic to our own tastes. The `\definetypeface` commands are not difficult to cut-and-paste and adapt to your own tastes.

You can see how Gill Sans is named as the sans member of the family, scaled down slightly to match Hoefler Text's x-height. Again, the encoding is abstracted from the typescript.

All of the font families are defined in similar ways to the above, but as the underlying fonts have different family members, there are more than a few exceptions and caveats in the mix:

- There are both `hoefler` and `hoefleroldstyle` typescripts defined. The regular one defines an oldstyle figures variant, and the oldstyle typescript defines a lining figures variant. Both define a small-caps variant. Variants are a way of switching font features beyond the usual regular/bold/italic alternatives.

An example of variant usage is to switch to SMALL CAPS *within a stretch of italic text with a grouped* `{\Var[sc] small caps}` *command.*

☞ For Didot, we have defined an additional oldstyle figures flavor, but no special `\Var[]` variants are made available because the oldstyle figures are only available from the regular face, not italic or bold.

Didot is of the same family as *Bodoni*, and shares some characteristics with *Computer Modern*. The old-style **numerals**, however, evince a very different design: 123 456 789.

☞ American Typewriter, although it has no italic, is available in Regular, Light, Condensed, and Light-Condensed varieties.

☞ Helvetica Neue is also presented as two families: regular and light. The light family uses the fashionable HelveticaNeue-UltraLight weight. It's not very suitable for text usage, but might make for interesting headlines.

American Typewriter, although a slab-serif, typewriter–style face, is classed more as a **serif** for ConTEXt use because it's not terribly well adapted for code printouts. Helvetica Neue is an artful re-drawing of a classic face that *nearly* outstayed its welcome. It has led to a resurgence and even *more* ubiquity, not only in **office** applications, but in **graphic design**.

☞ There are only four fonts within the Futura family shipped by default on the Macintosh: Medium, Medium-Italic, Condensed-Medium, and Condensed-ExtraBold. These don't make for a satisfying family, but we provided two (in despair): Futura and Futura-Condensed.

☞ Gill Sans, as already discussed, is available in light- and regular-weight families.

☞ Lucida Grande, because of its exceptional glyph coverage, has the `qx` (Central Europe) and `t5` (Vietnamese) encodings enabled within ConTEXt, simply because it was easy to do so.

I may not know much **tiếng Việt**, but I know my **glyphs**!

7 Symbols as a bonus feature in ConTEXt

As a surprise feature on the day before release, one of the authors dug up some old code and re-used it in order to enable use of the Hoefler Text Ornaments font. There are some attractive fleurons and borders within the font, and the authors thought it would be nice to have some basic support for users to experiment with. Only a couple commands are necessary within the `symb-gtahoefler` file, which takes the following form:

```
\loadmapfile[gtamacfonts]

\def\HoefO#1{\getglyph{HoeflerTextOrnaments}{#1}}

\startsymbolset[Hoefler Ornaments]

   \definesymbol[Hourglass]              [\HoefO{4}]
   \definesymbol[LeftHand]               [\HoefO{6}]
   \definesymbol[RightHand]              [\HoefO{7}]
   % ...

\stopsymbolset
```

With these definitions in place, the end user accesses the symbols in the Hoefler Text Ornaments font by loading the file (`\usesymbols[gtahoefler]`), loading the symbol set (`\setupsymbols[Hoefler Ornaments]`), and then calling the symbolic names with

commands such as `\symbol[Hourglass]`, and `\symbol[RightHand]`.

We know the font will be present if this symbol file is on a user's system, so all we need to do is ensure the `.map` file is loaded, define a convenience command, and name the symbols based on the names given directly in the encoding file. By combining these symbols, fairly pleasant effects (such as the acorn in the bulleted lists) can be created.

---

8 Packaging and Distribution

The results are organised according to the TEX Directory Structure (TDS). In our case, that means that there is a directory structure that can be grafted on top of an existing `texmf` tree. The various font-related files for this distribution are stored in `gtamacfont/` subdirectories for easy management:

```
./fonts/enc/dvips/gtamacfonts/
./fonts/map/pdftex/gtamacfonts/
./fonts/tfm/gtamacfonts/didot/ (etc)
./fonts/vf/gtamacfonts/didot/ (etc)
./doc/fonts/gtamacfonts/
./tex/latex/gtamacfonts/
./tex/context/gtamacfonts/
```

The gtamacfonts distribution has been made a part of the TEX **i-Installer**[8] i-Package. After installation and during configuration, the i-Package checks if in the `texmf.gwtex` tree, e.g. the directory `./fonts/truetype/gtamacfonts/didot` does not exist; if it doesn't, it creates it and populates it with an unpacked Mac Didot font using the `fondu` tool:

```
Didot.ttf
DidotBold.ttf
DidotItalic.ttf
```

[8] `http://ii2.sourceforge.net/`

To get this automatic conversion during the TEX i-Package configuration phase, it is necessary that the `fondu` tool has already been installed either manually or via the Fondu i-Package.[9]

9 Testing and Documentation

After installation with i-Installer, you can go to the

`/usr/local/teTeX/share/texmf.gwtex/doc/fonts/gtamacfonts/`

directory where you'll find several example files for LATEX and ConTEXt showing you how to use the fonts and which glyphs are available. The manual, `gtamacfonts.pdf`, is also available there.

10 Limitations and ToDo's

We hope that many users will find our support for Macintosh system-fonts useful, but we're also aware of a number of limitations, and there are a few features that we'd like to implement, but couldn't implement now for lack of time and skills:

1. The most important point is clear, yet it is well worth repeating: we are talking about TrueType fonts, so they will *only* work with pdfTEX, *not* with vanilla TEX and `dvips`. They will also be incompatible with packages and features that rely on PostScript features (such as `pstricks` or the font expansion parts of `microtype`). [*Character protrusion, however, is possible, and is activated in this document, as it is one of ConTEXt's standard features.*]

2. All fonts have some shortcomings. EC and texnansi define many characters; none of the fonts has them all. Some of the fonts are missing several weights and/or variants. Very few have additional features such as oldstyle figures and/or small caps.

[9] If `fontforge` is installed, the i-Package will also produce and store the AFM files.

3. Some fonts offer several weights such as light, ultralight, regular. In LaTeX, these could be made variants to the regular font, and one could even think of implementing switching to these alternative weights via options for the packages. This has not yet been done.

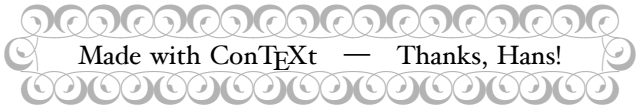
4. Both ConTeXt and LaTeX have a peculiar and not quite satisfying way of implementing support for oldstyle figures. In ConTeXt, it is either intrinsic to the font or enabled with a font variant call (such as `\Var[os]`). The old, and somewhat deprecated method of using the `\os` command is generally hard-wired into the MathItalic font and is a relic of some peculiarities of Computer Modern's old encoding conventions.

   In LaTeX, the command `\oldstylenums{}` produces oldstyle figures from the math font. Since we haven't redefined the math font, this would be Computer Modern, which doesn't look very good with these fonts. If the document uses the `textcomp` package, the command `\oldstylenums{}` takes the oldstyle figures from the accompanying `TS1` font if they are defined there. This is messy and will confuse inexperienced users. As a workaround, we recommend using `\textsc{}` in order to produce oldstyle figures (1234567890) where they exist and they are not default.

5. There is much more that could be done with the symbol support, both enhancing it within ConTeXt – for example, to take good advantage of the ornate border elements, there could be some enhanced pattern-generating macros – and to give basic support for LaTeX – we simply lacked the expertise to make it into a LaTeX package. We welcome any contributions in that area.

11 What the Future Holds...

We're quite happy that we could provide support for these fonts. We're looking forward to future developments. Now that the basic support is in place, it is fairly easy to add more fonts and to refine the support for the existing fonts. We also hope that our implementation will help reduce the wide-spread prejudice against TrueType fonts in TeX. And we hope that other users will feel welcome to join our efforts and contribute improvements.


Made with ConTEXt — Thanks, Hans!