A closer look at TrueType fonts and pdfTEX

Hàn Thế Thành

Abstract

Explanations and examples of using TrueType fonts directly with pdfTEX, especially the complications regarding encodings and glyph names.

1 Glyph identity in Type 1 vs. TrueType

The most common outline font format for TEX is Type 1. The TrueType format is rather different from Type 1, and getting it right requires some extra work. In particular, it is important to understand how TrueType handles encoding and glyph names (or more precisely, glyph identity).

We start with Type 1, since most TEX users are more familiar with it. In the Type 1 format glyphs are referred to by names (such as '`/A`', '`/comma`', and so on). Each glyph is identified by its name; so, given a glyph name, it is easy to tell whether or not a Type 1 font contains that glyph. Encoding with Type 1 is therefore simple: for each number n in the range 0 to 255, an encoding tells us the name of the glyph that should be used to render (or display) the charcode n.

With TrueType the situation is not that simple. TrueType does not use names to refer to glyphs, but rather so-called "indices": each glyph is identified by an index, not a name. These indices are simply numbers that differ from font to font. The TrueType format handles encodings by a mechanism called "cmap", which (roughly) consists of tables mapping from character codes to glyph indices. A TrueType font can contain one or more such tables, each corresponding to an encoding.

2 Glyph names vs. Unicode in TrueType

Because glyph names are not strictly necessary for TrueType, they are not always available inside a TrueType font. Given a TrueType font, one of the following cases may arise.

- The font contains correct names for all glyphs. This is the ideal situation and is indeed often the case for high-quality Latin fonts.
- The font contains wrong names for all or most of its glyphs. This is the worst situation that often happens with poor-quality fonts, or fonts converted from other formats.
- The font contains no glyph names at all. Newer versions of Palatino fonts by Linotype (v1.40, coming with Windows XP) are examples of this.
- the font contains correct names for most glyphs, and no names or wrong names for a few glyphs. This happens from time to time.

One may wonder how the situation can be so complex with glyph names in TrueType and still get anything typeset correctly. The reason is that Type 1 fonts rely on correct names to work properly. Thus, if a glyph has a wrong name, it gets noticed immediately. In contrast, as mentioned before, TrueType does not use names for encoding. So, if glyph names in a TrueType font are wrong or missing, it is usually not a big deal and can easily go unnoticed.

The potential problem with using TrueType in pdfTEX is that we TEX users are accustomed to the Type 1 encoding convention, which relies on correct glyph names. Furthermore, most font tools rely on this convention and all encoding files (`.enc` files) use glyph names. But, as explained above, glyph names in TrueType are not reliable. If we encounter a font that does not have correct names for its glyphs, we need to do some more work.

If glyph names are not correct, we need another way to refer to a glyph in TrueType fonts. The most reliable way seems to be via Unicode: usable TrueType fonts must provide a correct mapping from Unicode value to glyph index.

Since version 1.21a pdfTEX has supported the naming convention 'uniXXXX' in encoding (`.enc`) files. This makes sense only with TrueType fonts. When pdfTEX sees for example '`/uni12AB`', it

- reads the ⟨*unicode*⟩→⟨*glyph-index*⟩ table from the font, and
- looks up the value '12AB' in the table, and if found then uses the relevant glyph index.

The ttf2afm utility does the same lookup when it sees names like 'uni12AB'.

3 Using TrueType in pdfTEX

Let's review the minimal steps to get a TrueType font working with pdfTEX:

- Generate an afm from the TrueType font using ttf2afm. Example:

```
ttf2afm -e 8r.enc -o times.afm times.ttf
```

- Convert afm to tfm using any suitable tool — afm2tfm, fontinst, afm2pl, etc. Example:

```
afm2tfm times.afm -T 8r.enc
```

- Define the needed map entry for the font. Example:

```
\pdfmapline{%
  +times TimesRoman <8r.enc <times.ttf}
\font\f=times
\f Hello this is Times.
```

(The font name 'TimesRoman' used in the map line is declared inside the `times.ttf` file.)

The above deals with the easiest case: when glyph names are correct.

Now let us consider a font where we cannot rely on glyph names: Palatino version 1.40 from Linotype, for example. Let us assume that we want to use the T1 encoding with this font. So we put `pala.ttf` and `ec.enc` in the current directory before proceeding further.

First attempt:

```
ttf2afm -e ec.enc -o pala.afm pala.ttf
```

However, since the names in `ec.enc` are not available in `pala.ttf` — in fact there are no names inside this font — we get a bunch of warnings:

```
Warning: ttf2afm (file pala.ttf): no names
  available in 'post' table ...
Warning: ttf2afm (file pala.ttf): glyph 'grave'
  not found
...
```

and the output `pala.afm` will contain no names at all, but instead weird entries like 'index123'. Furthermore, glyphs are not encoded:

```
C -1 ; WX 832 ; N index10 ; B 24 -3 807 689 ;
```

We try again, this time without any encoding:

```
ttf2afm -o pala.afm pala.ttf
```

Since this time we did not ask ttf2afm to reencode the output afm, we get only the first warning:

```
Warning: ttf2afm (file pala.ttf): no names ...
```

but the afm output is the same as in the previous attempt. This is not useful, since there is little we can do with names like 'index123'.

So we try to go with Unicode:

```
ttf2afm -u -o pala.afm pala.ttf
```

This time we get different warnings, such as:

```
Warning: ttf2afm (file pala.ttf): glyph 108 has
multiple encodings (the first one being used):
uni0162 uni021A
```

At first sight it is hard to understand what ttf2afm is telling us with this message. So let us recap the connection between glyph name, glyph index and Unicode value:

- TrueType glyphs are identified internally by an index, not a name.
- ⟨*glyph-name*⟩→⟨*glyph-index*⟩ is optional, and the information may be wrong, if present. Likewise ⟨*glyph-index*⟩→⟨*glyph-name*⟩.
- ⟨*unicode*⟩→⟨*glyph-index*⟩, on the other hand, is (almost) always present and reliable.
- ⟨*glyph-index*⟩→⟨*unicode*⟩ is not always reliable, and need not even be a mapping, since there can be more than one Unicode value mapping to a given glyph index. That is, given a glyph index, there may be no corresponding Unicode value, or there may be more than one. If there is none, the glyph index will be used ('index123', for example). Now suppose that there are more than one, as in the example above, where 0162 and 021A are both mapped to glyph index 108

  In sum, we have asked ttf2afm to print glyphs by Unicode, and ttf2afm cannot know for sure which value to use. Hence it outputs the first Unicode value and issues the warning.

If all we want to do is to use `pala.ttf` with the T1 encoding, probably the easiest way is to create a new enc file `ec-uni.enc` from `ec.enc`, with all glyph names replaced by Unicode values. (This simple approach does not handle ligatures; see below.) This can be done easily enough by a script that reads the AGL (Adobe Glyph List, `http://www.adobe.com/devnet/opentype/archives/glyphlist.txt`) and converts all glyph names to Unicode.

Assuming that we have such a `ec-uni.enc`, the steps needed to create the tfm are as follows:

```
ttf2afm -u -e ec-uni.enc -o pala-t1.afm pala.ttf
afm2pl pala-t1.afm pltotf pala-t1.pl
pltotf pala-t1.pl
```

We could then use the font in pdfTEX as follows:

```
\pdfmapline{+pala-t1 <ec-uni.enc <pala.ttf}
\font\f=pala-t1
\f This is Palatino in the T1 encoding.
```

4 General solutions for fontinst et al.

If we want to do more than just using `pala.ttf` with T1 encoding, for example processing the afm output with fontinst for a more complex font setup, then we must proceed slightly differently. Having an afm file where all glyph names are converted to the 'uniXXXX' form, as we have done above, is not very useful for fontinst. Instead, we need an afm file with AGL names, do our processing, and then convert back to 'uniXXXX'. We can do this as follows.

- Generate the afm with 'uniXXXX' glyph names:

  ```
  ttf2afm -u -o pala.afm pala.ttf
  ```
- Convert that `pala.afm` to `pala-agl.afm`, so that `pala-agl.afm` contains only AGL names. A script similar to the one mentioned above can do this.
- Process `pala-agl.afm` with fontinst or whatever else is desired.
- In the final stage, when we have the tfm's from fontinst (et al.), plus the map entries (from fontinst or created manually), we need to replace the encoding by its counterpart with the

'uniXXXX' names, since that is what the actual TrueType font requires. For example, if fontinst tells us to add a line saying

```
pala-agl-8r <8r.enc <pala.ttf
```

to our map file, we need to change that line to

```
pala-agl-8r <8r-uni.enc <pala.ttf
```

where `8r-uni.enc` is derived from `8r.enc` by converting all glyph names to the 'uniXXXX' form.

The encoding files distributed with the TeX Gyre fonts cover just about everything a typical TeX user needs. Those encodings have been converted to the 'uniXXXX' form for your convenience and are available at `http://tug.org/fontname`, with names such as `q-ec-uni.enc`.

5 Disappearing glyphs and final tips

Another problem that happens from time to time is being sure that a glyph exists inside a font but we don't get that glyph in the pdfTeX output.

The likely cause is the glyph being referenced by different names at the various stages the process of creating support for the font, e.g., the tfm, vf, enc and map files. For example, the names 'dcroat', 'dbar', 'dslash' and 'dmacron' can all refer to the same glyph in a TrueType font. In general, the origin of a glyph name can come from several sources:

- the individual font itself;
- a predefined scheme called "the standard Macintosh ordering of glyphs" (unfortunately the TrueType specifications by various companies (Apple, Microsoft and Adobe) are not consistent in this scheme and there are small differences, for example 'dmacron' vs. 'dslash');
- the result of the ⟨*unicode*⟩→⟨*glyph-name*⟩ conversion, according to the AGL.

In such situations, probably the easiest and most reliable way to get the glyph we want is to use a font editor like FontForge (`http://fontforge.sf.net`), look into the font to discover the specified Unicode for the glyph and then use the 'uniXXXX' form to instruct ttf2afm and pdfTeX to pick up that glyph.

Finally, another way to get a problematic TrueType font to work with pdfTeX is to forget all of the above and simply convert the font to Type 1 format using FontForge. While it sounds like a quick hack, there is nothing necessarily wrong with this; it can be a simple and effective workaround.

◇ Hàn Thế Thành
River Valley Technologies
`thanh (at) river-valley dot org`