*Opening up the type**

TACO HOEKWATER
Elvenkind, Dordrecht
taco (at) elvenkind dot com

Abstract

In the near future, pdfTEX will gain the ability to use OpenType fonts. This paper explains in broad terms what will be done and why it will be done in that way.

Introduction

There are many reasons why OpenType support is a good thing for pdfTEX to have. For one, many of the latest commercial fonts can only be purchased in OpenType format. For another, most of the new OpenType fonts are of higher quality than their older brethren. Also—this is a minor point, but not completely unimportant—everybody else does it. So long as pdfTEX does not support OpenType, it has no chance of being perceived as a viable competitor to those other systems.

Finally and perhaps most importantly, OpenType allows us to escape from TFM format and its many limitations, without the need to invent another special font format that can only understood by TEX and not much else.

The plan is for the OpenType support to be included as part of a project to create a high-end Arabic typesetting engine based on a merge of pdfTEX, Aleph and luaTEX. The final result of this project will be open source and can be merged into future versions of pdfTEX.

Wishes and constraints

Of course we want to integrate the new OpenType support such that it behaves well with the rest of TEX:

- It should be possible to use all of the glyphs and features in the font.
- The implementation, its input, and its output should all be platform independent: same font, same syntax, same typesetting.
- We want to make the implementation extensible, just in case someone comes up with OpenType++ in the next decade.
- It should still be possible to define virtual fonts and use the current pdfTEX micro-typography features like protruding and font expansion.
- The ability to make run-time adjustments to the font characteristics is desirable.
- The interface should be usable by somebody who is not trained as a programmer.
- For nice and small pdf output, some sort of font subsetting has to take place.
- Backward compatibility code for traditional TEX and PostScript fonts has to remain; the goal is evolution, not revolution.
- We wish full control at the basic glyph level, not limited to turning OpenType features on or off.

Unicode

When dealing with OpenType fonts, adding support for Unicode is more or less implied. Therefore pdfTEX had to be extended to handle Unicode input as well as output.

File I/O

Partial Unicode support is already in the current luaTEX code base:

- UTF-8 encoded text input and output.
- Characters and tokens use 21 bits for storing character information.
- Hyphenation patterns are loaded in UTF-8.
- The pool file (strings) and buffer are Unicode enabled.

*This work is made possible by a grant from Colorado State University.

Characters

At present, TEX does not have a concept of a character: a 'charnode' is actually a glyph in a font, together with the font it should be taken from.

To fix this unpleasant intermingling of glyphs with characters, pdfTEX will be extended with two new node types:

- A font node is the result of a font selection command by the user.
- A unichar node is the result of tokens being read.

The 'charnode' functionality is still present (renamed to glyph node), but the new types will not be converted to the traditional TEX glyph, font pair until after the hyphenation pass is completed.

Hyphenation

The old node list and paragraph building routines intertwined ligature building, hyphenation and line breaking. On top of that, hyphenation patterns were stored using the 'charnode', as mentioned in the previous paragraph.

This resulted in a few unfortunate side-effects:

- patterns are font encoding dependent;
- hyphenation is impossible unless a `\hyphenchar` is present in the current font;
- hyphenation patterns can only use 256 characters at a time.

The new code separates hyphenation from the line breaking decisions: First it finds *all* potential hyphenation points in the words (made up of unichar nodes) and insert `\discretionary` nodes for all of them. Only after that step is completed will it attempt to find ligatures and break the paragraph into lines.

This change makes hyphenation completely independent of the current font.

A different internal representation of the loaded patterns will make it possible to use the full range of Unicode characters in hyphenation patterns as well as making it possible to extend the patterns in a language at run-time.

Languages

We believe this is a good opportunity to also tackle another traditional problem in TEX: the `\lccode`, `\uccode` and `\sfcode` tables. These tables contain information that is conceptually part of the current language, and should not be stored in a font attribute.

We want to increase the importance of `\language` codes and attach much more information to language switches. Other candidates for inclusion in language switching are the `\uchyph` parameter and the list of applicable ligatures.

Scripts

pdfTEX currently uses the TEX–XET algorithm from ε-TEX, with the primitives `\beginL` and `\beginR`.

This will be removed in favor of the much more advanced and flexible Aleph/Omega1 typesetting direction commands `\pagedir`, `\pardir`, etc.

An equivalent to ΩTP processing will be implemented using lua instead of ΩCP (precompiled binary) files.

Font loading

In current pdfTEX, fonts are internally represented as a large storage heap with a few dozen auxiliary tables that store various meta-information and pointers into the heap. All of those are global, and implemented as static objects.

While this is very efficient in terms of speed, it is also very hard to alter a font after it has been loaded, and the unification forces all fonts to offer strictly the same interface.

In the new setup, fonts will be loaded under the direct control of lua code, and they will be presented to the typesetting engine as a single lua table for each loaded font. This table will make the font behave much like an object that can be queried and altered directly by the macro programmer, either from TEX macro code (through `\fontdimen`) or from lua code (through callbacks from the typesetting engine).

The low-level font loading routines will be written in compiled C code, perhaps by using a separate library like freetype.

Conclusion

We gratefully acknowledge that this work is made possible by a grant from Colorado State University, with the sponsorship of Idris Hamid, and with support from TUG. A test version of the changes described in this paper should be available before the TUG 2006 meeting in Morocco. People wishing to stay up to date with respect to this project are invited to visit `http://www.luatex.org`.