Attributes: The New Kid on the Block

When you switch fonts in TEX grouping keeps changes to another font local to the group. But there is more than fonts. Most macro packages provide color support and in ConTEXt we also support some PDF related features like outlines and hidden invisible ink. These features all share a common problem: we need to keep track of their state in the page stream and across pages and splitting content also takes some care. A maybe less obvious example are hyperlinks, which is natively supported by pdfTEX (although ConTEXt does it slightly differently).

In order to make such features easier (and more robust) to implement luaTEX provides attributes. These behave like fonts but are by design indifferent of what they represent. They travel with the nodes (each node can have attributes) and it's up to the macro package to make sure that the intended behaviour takes place. For this Lua code is used in combination with processing node lists, either by using callbacks or by postprocessing boxes.

In this talk I will explain what attributes are, and how they can be of use to macro writers.

ConTEXt MkIV: luaTEX hits the road (could be talk and tutorial)

In ConTEXt dealing with different backends has never complicated the system because driver dependent features are isolated rather well and use a driver independent layer. Different frontends are supported by conditional sections in the code which are dealt with at format generation time. When XƎTEX came around, frontend issues became more noticeable and with luaTEX we can safely say that we're dealing with a new kind of TEX (although it is still downward compatible with pdfTEX).

In order to use luaTEX to its full power, i.e. go beyond what pdfTEX provides, one needs to extend of even rewrite substantial parts of macro packages. As we develop luaTEX, Taco and I do lots of tests and we use ConTEXt as a testbed. In the process large parts of this macro package are replaced and the related version is tagged MkIV (its precedecessor has tag MkII). When luaTEX has become stable we will rewrite parts of ConTEXt even more drastically but the current state already gives a good impression of the impact that luaTEX will have on writing macros.

In this talk I will discuss what impact luaTEX has on the current releases of ConTEXt, and what the consequences will be for its future.

Zapfino: Hermann's Torture Test for TEX

The next couple of years TEXies have to explore the new landscape of OpenType fonts. Most of the implementation details will be hidden beyond user interfaces of macro packages. However this does not hide the potential mess that users can invoke when they start enabling or disabling features related to fonts.

Thanks to the Oriental TEX project Taco can spend substantial time on coding luaTEX which in turn means that I have lots of testing and protyping on my plate. We also spend much time on discussing the interfaces and extensions to the program and due to this as well as realistic testing luaTEX develops rapidly. However, in order to fulfill the requirements of the Oriental TEX project we need to be able to typeset high quality Arab. Since I'm more familiar with Latin and since I had the Zapfino Pro handwriting font waiting for me in OpenType format I decided to use that font as benchmark for advanced node processing in luaTEX. It proved to be a worthy contender. In the process we were able to optimize node support in luaTEX and it also triggered reimplementing the OpenType tables (from FontForge format 1 to format 2).

In this presentation I will discuss how we deal with advanced features that are part of OpenType fonts like Zapfino. I will also explain how such features are implemented in Lua and TEX code.