Designing an implementation language for a TEX successor

David Kastrup
dak@gnu.org

Abstract

Managing the complexity of TEX's codebase is an arduous task, so arduous that few mortals can hope to manage the underlying complexity. Its original author's computational roots date back to a time where the maturity and expressive power of existing programming languages was such that he chose to employ the assembly language of a fictional processor for the examples in his seminal work "The Art of Computer Programming". In a similar vein, TEX is written in a stripped-down subset of a now-extinct Pascal dialect. Current adaptations of the code base include more or less literal translations into Java (NTS and exTeX), C++ (the Omega-2.0 codebase), mechanically generated C (web2c) and a few others. In practically all currently available cases, the data structures and control flow and overall program structure mimick the original program to a degree that again requires the resourcefulness of a highly skilled programmer to manage its complexity. As a result, almost all of those projects have turned out to be basically single-person projects, and few projects have shown significant progress beyond providing an imitation of TEX.

It is the persuasion of the author that progressing significantly beyond the state of the art as represented by TEX will require the expressiveness and ease of use of a tailor-made implementation and extension language. Even a language as thwarted as Emacs Lisp has, due to its conciseness, rapid prototyping nature, extensibility and custom data types and its coevolution with the Emacs editor itself, enabled progress and add-ons reaching far beyond the original state as conceived by its original authors. This talk tries to answer the question what basic features an implementation platform and language for future typesetting needs should possess.