♢ ♢ ♢

# *A Tour around the $\mathcal{NTS}$ implementation*

Karel Skoupý

ABSTRACT. $\mathcal{NTS}$ is a modular object-oriented reimplementation of TEX written in Java. This document is a summary of a presentation which shows the path along which the characters and constructions present in the input file pass through the machinery of the program and get typeset. Along the way the key classes and concepts of $\mathcal{NTS}$ are visited, the differences with original TeX are explained and the good points where to dig into the system are proposed.

KEYWORDS: $\mathcal{NTS}$, Java, extension

$\mathcal{NTS}$ is a modular object-oriented reimplementation of TEX. It is written in Java and is meant to be extended with new functionality and improvements. In spite of the expectations of many it is not simpler than original TEX and it probably could not be if it has to do exactly the same job. But whereas TEX is a very monolithic system, the complexity of $\mathcal{NTS}$ is divided into many independent modules and is accommodated in lots of useful abstractions. As a consequence one need not know all the details about the whole system in order to extend or change just a specific part. The dependencies between the modules are expressed by clear interfaces and the interfaces is all one needs to know about untouched parts. So extending and changing $\mathcal{NTS}$ should be quite easy, shouldn't it?

The problem is that detailed documentation is still missing and that there are hundreds of classes, so it is hard to know where to start. Although the classes are many, fortunately there are a limited number of main concepts which are really important.

The processing in $\mathcal{NTS}$ is naturally quite similar to TEX. The input is scanned and converted to *Tokens*. Each *Token* has a certain meaning: a *Command*. Some *Commands* are non-typographic, these usually deal with macro expansion or registers. Typographic *Commands* build some typographic material consisting of *Nodes* using *Builders*. Lists of *Nodes* are packed by a *Packer* and finally typeset by an instance of *Typesetter*.

Of course there are a few more basic concepts than those emphasized in the previous paragraph but not that many. They are always represented by an abstract class or an interface. The other classes in $\mathcal{NTS}$ are either various implementations of those interfaces or they are auxiliary and not so interesting.

Recently we have been trying to improve the design of $\mathcal{NTS}$ so that extensions and configuration are even easier. We will also look into ways how to increase performance and interoperability with the TEX directory structure.