# Abstracts without papers

## Introducing Continuous Integration (CI) to TeX binaries

*Mojca Miklavec*

TeX Live binaries are being built once per year for about 20 different platforms by a number of volunteers and never get updated during the year. This is a good compromise between users' demand for reasonably new binaries, stability and the burden on volunteer builders and packagers.

ConTeXt community on the other hand strongly depends on the availability of the latest binaries of LuaTeX at any given time. There are also occasional requests for the latest binaries of XeTeX when new features get implemented.

We have recently set up a build infrastructure that can automatically build TeX binaries after every commit for a number of platforms, send emails when builds break, show reports and make the binaries available to users.

This approach puts a lot of burden off the shoulders of people previously responsible for building TeX binaries while at the same it time gives us freedom to run the builds a lot more frequently, getting binaries to users much faster and providing earlier feedback about problems to developers.

## Updates and TODOs “Cewe/XML to PDF” photobook conversion

*Harald König*

my TODOs for the photobooks:

- JPEG tooling (EXIF orientation/rotation)
- hue/staturatoin changes on JPEGs (background images)

## Vexillography in ConTeXt

*Marek Treťák, Tomáš Hála*

Vexillography is the part of vexillology that deals with drawing flags and banners. The common way how to use flags in a document is to download ready files, sometimes in vector format, sometimes as bitmaps, but there is no tool available for simple drawing in ConTeXt.

This is the reason why the ConTeXt module for drawing facilitation has been prepared. The module provides not only the basic set of flags and banners of European countries but brings the tool for defining own flags or banners which will be shown using some examples.

The talk will also cover basics of terminology and the most frequently used patterns of flags.

## Rules

*Taco Hoekwater and Hans Hagen*

When Mojca asked about adding rules to the end of lines, Taco offered to provide some insight in tricks to achieve this using Lua. This is also an anchor for explaining how the linebreak algorithm works and what eventually comes out.

As a follow up Hans implemented a mechanism in the core (in fact it was mostly an extension of an existing mechanism) and used the opportunity to extend some related mechanisms as well. Of course we also kick in some MP code (for Alan).

## Setups

*Wolfgang Schuster (and Hans Hagen)*

The user interface is described in xml files but the actual descriptions lagged behind development. Wolfgang spent a considerable amount of time to describe all commands in detail and we updated the descriptive format in the process. The rendering was partly redone, as was the help system and scripts that use this information.

## Scripts

*Hans Hagen (optional)*

Most users will only run the mtxrun and context scripts (and maybe the font one) but there are few more. I will give an update in what there is and what they are used for: how they help me and how they can help you.

## Workflows

*Hans Hagen with everyone*

What problems do we face when we integrate ConTeXt in a workflow and how can we deal with them. (Follow up on previous topic.) What more is needed.

## Rendering math

*Hans Hagen*

Now that the OpenType specification explicitly mentions math the renderer can be improved. One of the problems has always been that the fuzzyness of the specification resulted in all OpenType math fonts doing things slightly different. There is no way an engine can deal with this so either the fonts need to be improved (what happens indeed) or we need ways to manipulate them. Some examples will be given.

## Columnsets redone

*Hans Hagen*

From MkII we inherited two column handlers: a mechanism that could mix single and multi colummn mode, and a more rigid columnsets model. Both are still present, but replaced by mixed columns and pagegrids. Eventually the old models will be removed from the core (and become modules) as the new ones can perform better (and can still be extended). I will discuss some of the problems we face and solutions provided.

## Combining the power

*Hans Hagen*

The TeX, MetaPost and Lua languages each have their charm and strength and in ConTeXt we bring them together. In this presentation I'll give an example (or maybe a few more) about where this integration happens and what makes me decide which language to use for what aspect of a solution. (Stepcharts)

## Piece of Snake

*Taco Hoekwater*

How to implement snake-justification in ConTeXt, or: how to automatically fill in the ragged borders of non-justified paragraph text using embellishments. See `http://xkcd.com/1676/` for the inspiration.

## A short history of punctuation

*Taco Hoekwater*

Punctuation is "the use of spacing, conventional signs, and certain typographical devices as aids to the understanding and correct reading, both silently and aloud, of handwritten and printed texts." (Encyclopaedia Brittanica).
Punctuation evolved over time, just like everything else related to writing. This talk gives a short overview of the development process until now.

## CAKE: Source overview

*Taco Hoekwater*

ConTeXt Advanced Knowledge Essentials: Knowing where the various functionalities of ConTeXt are found in the source tree is helpful in (almost a prerequisite to) getting better acquainted with advanced functionality. This is an overview of what is where in the ConTeXt source after the rewrite for MkIV.

## CAKE: The TUC file

*Taco Hoekwater*

ConTeXt Advanced Knowledge Essentials: The temporary file ConTeXt uses to maintain state between consecutive runs of the typesetting engine contains lots of important information, but it is not easy to interpret by a novice. We will have a look at all the various objects contained in the temporary file, and how it can help to deepen your understanding of ConTeXt.

# ConTEXt Source Code Consistency Checker

*Adam Hanuš, Tomáš Hála**
** 1. Department of Informatics, Faculty of Business and Economics, Mendel University, Brno, Czech Republic; 2. KONVOJ, spol. s r. o. (publishing house), Brno, Czech Republic; Email addresses:* `thala@mendelu.cz, konvoj@konvoj.cz`

The compilation of a source code written in ConTEXt is understandably more time demanding. Therefore, each syntactic mistake unnecessarily delays the user on his way to the final version of his document. For improving the user's comfort, the consistency checker has been implemented. The checker makes it possible to detect the following syntactic offences even before the compilation starts:

(a) unbalanced braces and plain commands for opening and closing of groups;
(b) unbalanced start/stop command;
(c) check of the proper number of parameters;
(d) warning in front of undesirable spaces in key/values definitions.

The set of "pair" commands can be extended by the user editing the configuration file. Additionally, the check of included files and unbalanced parenthesis and brackets can be activated by options.
The consistency checker of ConTEXt source codes has been implemented in language Lua as a standalone programme. The use of the list of commands taken directly from ConTEXt system sources for substituting a part of configuration file is under development.

ConTEXt, syntax, consistency check, Lua

# TypoChecker: Checking and Correcting Selected Typographic Phenomena

*Dominik Makeš, Tomáš Hála**

** 1. Department of Informatics, Faculty of Business and Economics, Mendel University, Brno, Czech Republic; 2. KONVOJ, spol. s r. o. (publishing house), Brno, Czech Republic; Email addresses:* `thala@mendelu.cz, konvoj@konvoj.cz`

For simplifying proofreaders', typesetters' and editors' work, the programme for checking selected typographic phenomena has been prepared.
It is not always possible to decide unambiguously where exactly the author made a mistake. Therefore, the checker does not operate automatically; rather, the process has been divided into two stages. First, mistakes are only detected and the well arranged log file with detail description of mistakes is prepared for the user. The log file also contains proposals how to fix the detected mistakes. Then, on user's demand, all mistakes are corrected in mass. In this way the user can, by editing the file with the list of errors (especially by erasing records of points which should not be changed), influence the subsequent correcting process.
The presented version stems from Czech and Slovak typesetting rules. Additional rules reflecting typesetting rules in other languages, will be incorporated in the future.
The checker has been implemented in language Lua as a standalone programme.

ConTEXt, typochecking, detection of mistakes,
correction of mistakes, Czech, Slovak, Lua

## Implementation of Tokeniser Based on Lua Regular Expressions and Its Use for Highlighting the Syntax

*Tomáš Hála*
*1. Department of Informatics, Faculty of Business and Economics, Mendel University, Brno, Czech Republic; 2. KONVOJ, spol. s r. o. (publishing house), Brno, Czech Republic; Email addresses:* `thala@mendelu.cz, konvoj@konvoj.cz`

For some simple cases it is not necessary to write a new or complex tokeniser. Highlighting the syntax very often belongs to these simple cases.
Therefore, the module LexAn has been prepared. The module can recognise tokens which are described by regular expressions. The module has been implemented in language Lua, so we talk about reduced regular expressions. Tokens are marked with the corresponding keys, and these can be used for subsequent operations.
The presented solution can be used not only for highlighting the lexical elements by pre-defined set of regular expressions which is the usual situation, but, first of all, makes it possibe to define own rules (own sets of lexical symbols). Then, one can mark any part according to his/her needs, which could be used with a great advantage, e.g., for teaching.
The predefined sets of regular expressions describe programming languages Pascal, TEX, lua, perl and PHP. Development of other sets is in progress.

*Supported by CSTUG.*

ConTEXt, tokeniser, regular patterns, highlighted syntax, Lua