ConTEXt: a general-purpose TEX macro package

Giuseppe Bilotta

Department of Mathematics,
University of Catania,
Italy
E-mail: `gip.bilotta@iol.it`

Abstract

This article discusses ConTEXt, a powerful yet not very well known TEX format strongly oriented towards non-mathematical typesetting.

1. Introduction

The name of the game

ConTEXt is a TEX format developed at PRAGMA-ADE (`www.pragma-ade.com`) by Hans Hagen and a small team (the ConTEXt Task Force). It was initially developed to provide an enhancement of LATEX's itemization features and a localized (Dutch) wrapper to TEX. The package has now developed in a full-blown, independent TEX format, featuring an object-oriented, parameter-driven interface and a huge set of capabilities. The localized interface has not been abandoned, and the format is now available with commands in either English or German, Dutch, Italian, Romanian, Czech. Other interfaces are possible, provided a maintainer accepts to take over the effort of translating the commands and messages and keeping them up to date.

Indeed, an important characteristic of ConTEXt is that it is somewhat still a work-in-progress: while the core set of features is now rather stable (except for rare major overhauls, as it happened recently for the font interface), new features are continuously developed and added to the system, be them feature requests from the users (Hans is very responsive on this) or just Hans Hagen's own ideas of what would be nice to have.

ConTEXt is also projected towards support for modern standards; it has complete support for PDF production, including extensive interaction-oriented features, and native XML parsing capabilities. To achieve this, it fully exploits the enhancements to Knuth's TEX as provided by ε-TEX and pdfTEX, possibly combined in pdf-ε-TEX. While pdfTEX is not needed for PDF production (it can still be used for the other micro-typography enhancement not found in Knuth's TEX), XML parsing is only available when an ε-TEX-based compiler is used. Overall, it is strongly suggested to use the latest pdf-ε-TEX when using ConTEXt, to get the most out of it, at the least price (there are some code optimizations used when ε-TEX is the compiler engine).

Finally, for the graphics stuff, ConTEXt comes with its own MetaPost format, called MetaFun, a powerful toolbox whose most interesting feature is probably the tight integration with ConTEXt itself, which allows some interesting tricks combining TEX powerful typesetting capabilities and MetaPost's graphic power.

2. Sampling ConTEXt

As it was mentioned in the previous section, ConTEXt is an object-oriented format. TEX users familiar with LATEX know that in Lamport's format each logical document element is considered an "environment": itemizations, theorems and proofs, equations in their various forms, and the document itself; ConTEXt uses a similar approach, except that such structured convention is not imposed on the user —free form ConTEXt documents are possible: but they are not used very often, since part of the ConTEXt power depends on the internal knowledge of the structure. So, a "Hello, world!" ConTEXt document will look like this:

```
\starttext
Hello, world!
\stoptext
```

Before the first `\starttext` (which takes the role of LATEX's `\begin{document}`) one can set up various elements of the document: the paper and page size, how would the headings look like, etc. Such customizations can be collected in separate files, which are called "environments" in ConTEXt speech (note the difference with LATEX: in Lamport's format, environments are the document logical building blocks; in ConTEXt an environment is a document setup file which can be shared between documents —something alike to LATEX's classes and packages. LATEX's environments are called start/stop pairs in ConTEXt.)

LATEX has a very modular design: it provides a basic kernel and some default classes, and anything which deviates from standard has to be provided by (a bunch of) separate packages. On the contrary, ConTEXt uses the monolithic approach: most features are provided in the core, and customization is achieved via the parameter-driven interface. (ConTEXt does support the use of modules, though.) An example of such customization is the following snippet:

```
% we set up the page and paper size of our choice:
% note that page size can be different from paper size,
% preparing the ground for page composition
\setuppapersize[A4][A4]

% we set up the page numbering style:
\setuppagenumbering[alternative=singlesided,
                    location={footer,middle}
                    way=bytext,partnumber=no]

% next, we enable colour output
\setupcolors[state=start]

% and interaction
\setupinteraction[state=start,
                  color=blue,
                  style=normal]
```

As it can be seen, the key/value pairs can be entered in a very free-form layout, to the benefit of readability. An important bonus which comes from the key/value approach is that single characteristics of each configurable element can be changes without touching the overall look: for example, in the previous excerpt it can be seen that the color of the active elements can be changed separately from the font style; similarly, when changing a heading style, one may choose to only change the font shape or style or color (any or all of them) of the text or of the number (or both), while leaving the layout as-is, or to change selected parts of the layout (like: wether to start a new page or not, or wether to force an even or odd page, etc).

In most cases there are some predefined alternatives, accessible via the `alternative=` keyword. An example of this can be seen in figure 9, whose originating code is:

```
\environment con-ex
\starttext
\showframe
\setuphead[section][alternative=normal]
```

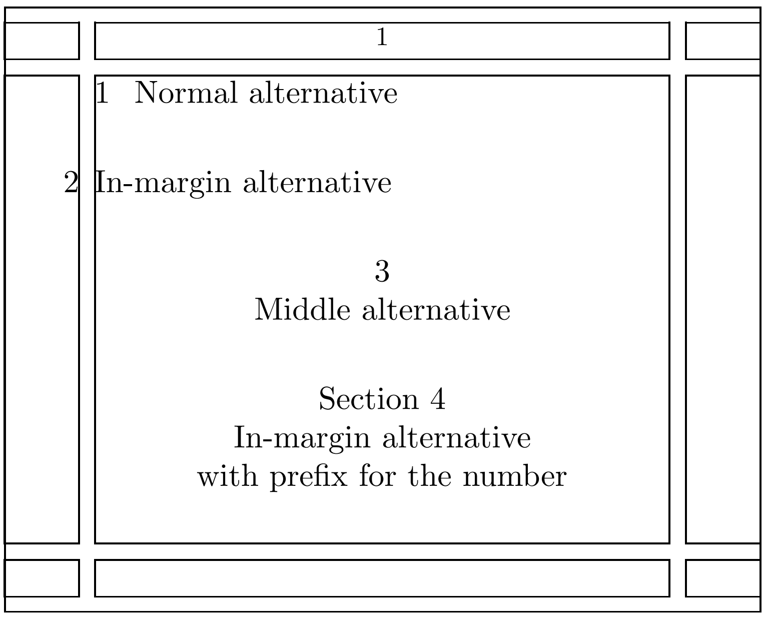

Figure 9: Some examples of default title layouts

```
\section{Normal alternative}
\setuphead[section][alternative=inmargin]
\section{In-margin alternative}
\setuphead[section][alternative=middle]
\section{Middle alternative}
\setuphead[section][numbercommand={Section~}]
\section{In-margin alternative\\ with prefix for the number}
\stoptext
```

An interesting feature demonstrated by this example is the `\showframe` command, that shows all the elements of the layout definition; an important feature of ConTEXt is indeed the presence of these "debugging commands", helpful to track layout setups, reference marks, grid placement, etc; debugging commands are also present in MetaFun, ConTEXt's MetaPost counterpart, to show control points, path directions, etc.

The environment file (shared by all compiled examples in this article) defines and activates a custom paper size appropriate for virtual pages which can become figures, sets up an appropriate custom layout and reduces the font size (the default is twelve points).

```
\startenvironment con-ex

\definepapersize[EX][width=10cm,height=8cm]
\setuppapersize[EX][A4]
\setuplayout[margin=1cm,margindistance=.2cm,
             header=.5cm,footer=.5cm,
             headerdistance=.2cm,footerdistance=.2cm,
             topspace=.2cm,bottomspace=.2cm,
             topdistance=0cm,bottomdistance=0cm,
             width=fit,height=fit]

\setupbodyfont[10pt]

\stopenvironment
```

3. Using ConTEXt

To gain the most out of ConTEXt it is important to have a Perl environment available; the ConTEXt distribution contains indeed a couple of Perl script, some of which are essential to exploit some features.

Graphics inclusion, for example, relies on a Perl script (`texutil.pl`) to determine the default size of most common binary formats; this same Perl script post-processes the auxiliary files for things like sorting index entries and manage other information which gets passed between one run and the next: like for LATEX, ConTEXt complex documents need multiple passes (if not else to resolve cross references). A Perl script (`texexec.pl`) takes care of running ConTEXt enough times, with the necessary intermediate `texutil.pl` runs, and possible MetaPost runs which may be needed to compile embedded MetaPost code.

(As it was mentioned before, ConTEXt integrates tightly with MetaPost. A ConTEXt document can contain MetaPost code which will be output to temporary files which will be compiled before the next ConTEXt run. Enabling the `\write18` hack available in most recent distributions will allow ConTEXt to run MetaPost in a subprocess, thus reducing the number of intermeditate runs needed.)

There are also some auxiliary Perl scrips, like `texfont.pl` which takes care of creating all the necessary support files needed to use non-default and non-standard fonts, or `texshow.pl`, a Perl/Tk script that displays a summary of each ConTEXt command.

4. Advanced ConTEXt

4.1. Versions and modes

As it was mentioned before, ConTEXt comes with a built-in set of features oriented towards interactive (PDF) document production. To summarize these briefly, there are screen-sized predefined "paper" size, hyperlink capabilities, tooltips and JavaScript support, movie/sound playback support, forms/fields capabilities, some predefined presentations styles (which also function as examples for some of the less-documented ConTEXt features).

The same document can function as a base for both an interactive (screen) and a printed (paper) version of the same document (most ConTEXt documentation itself, and the pdfTEX manual as well, make use of this feature). This effect is achieved via the use of "modes": the parts of the document which should only be compiled when producing a certain version of the document (or when *not* producing a certain version) can be surrounded by `\startmode[mode name(s)]` ... `\stopmode` (or `\startnotmode[mode name(s)]` ... `\stopnotmode` respectively.)

Another important application of these feature is, for example, to prepare school textbooks, with different versions for the teacher and for the students (the stutent's book could for example contain the exercises without the answer, which would be present in the teacher's book.)

To make use of this feature, the `--mode mode name(s)` option can be passed on to the `texexec.pl` compiler script. (As a side note, other common options to this script are `--pdf` which triggers direct PDF production when a pdfTEX-based compiler is used, or `--color` to enable colours —this last option is equivalent to `\setupcolors[state=start]` in the document.)

ConTEXt also treates draft and final versions of the same document in a different manner: the user can specify which version the document with the command `\version`, which accepts either `final` (the default) or `temporary` or `concept` as a parameter. When the document is not in its final shape, additional debugging information is shown.

4.2. Buffers, outer world interface

A typical need in preparing TEXnical documentation is the need to use the same piece of text multiple times (for example, once verbatim and once in compiled form to show the resulting output).

ConTEXt has built-in capabilities to deal with reusable blocks. The feature can be used in different ways. A classical example is the one mentioned above; another common use is showing different language-specific typesetting defaults (quoting styles, date formats, punctuation spacing, etc); or to provide for delayed use of pieces of text (a school book can for example have the exercises set up right after the topic they refer to, but typeset only at a later stage; or, answers can be written down right after the exercises, but typeset at the end of the document in an appropriate appendix; this can be combined with the modes feature to provide different layout/order for the teacher and pupil versions.)

Finally, this same feature is used to communicate with the outside world (especially to pass on information to MetaFun)

4.3. Master documents

While it is possible to use TEX's native `\input` command to split documents into smaller parts, ConTEXt comes with some built-in stuff for master/sub document management. The format assumes an (at most) three-levels deep structure: projects, products, components.

A typical usage of this feature can be for example the compilation of a set of manuals with a consistent setup (project); each manual would be a product, and the single chapters could be the components. (An alternative approach is to define the project to be the single manual, the parts to be the products, and the single chapters the components.)

Using this feature one can save (compilation) time, by only compiling the component/product (s)he is actually working on, and only compiling the product/project when a global vision of the whole work is needed.

4.4. Page arranging

When using pdfTEX-based compilers, ConTEXt offers page imposition and rearranging, achieved through the native pdfTEX support for inclusion of single pages from a PDF document, with optional scaling and rotation. In this sense ConTEXt offers capabilities similar to those found in the PSUtils bundle, for PDF files; it is to be noted that they work for any PDF document, and not just for ConTEXt-producted PDF.

At least two runs are always needed to obtain the rearranged/merged version of a document: one for the creation of the document and one for the actual imposition. If the document source contains page arranging istructions, `texexec.pl` will take care of the whole job, by doing the double runs itself. Otherwise, it is always possible to specify page arranging options on the `texexec.pl` command line. For maximal control it is better to create a “shell” document containing the specificications; personally, I use this core:

```
% format=english
\setuppapersize[A4][A3,landscape]
\setuparranging[2UP]
\setuplayout
  [topspace=0pt,
   backspace=0pt,
   header=0pt,
   footer=0pt,
   width=middle,
   height=middle]
\starttext
\insertpages[atd.pdf]
\stoptext
```

adapted as needed.

5. ConTEXt vs LATEX

The basic feature set found in ConTEXt is essentially the same as that provided by any LATEX standard distribution. Preference of one system over the other is therefore dependant of the advanced use one wants to achieve.

On the ConTEXt side one can consider the following advantages:

— ConTEXt is monolithic: the plethora of LATEX packages needed to change the default look of the standard LATEX classes or do add advanced features can be confusing for the beginner, especially considering that many packages have similar functionality or provide overlapping features. Also, while most standard packages cooperate fairly well, some of the more obscure or outdated ones can break havoc without warning, and finding the culprit (or a solution) is not always easy. ConTEXt on the other hand provides a consistent, robust, internally compatible set of features, most of which are available right out-of-the-box.

— ConTEXt is customizable. Almost any aspect of a ConTEXt document can be easily controlled and fine tuned. This may not be very important for technical books, but less restricted environments like the humanities can take advantage of this (the recently developed “memoir” and “newlfm” LaTeX classes try to bridge this gap, but they still need extra packages to provide the complete set of features offered by ConTEXt.)

— ConTEXt is powerful and its feature set is constantly growing. ConTEXt comes with built-in poweful and flexible features: columns, column sets (for DTP-like effects), various tabular and table types, interactive features, native XML support, MetaPost integration. All these feature work nicely with each other thanks to the integrated, monolithic approach.

— ConTEXt tries to take full advantage of post-Knuthian TEX variants (pdf-ε-TEX up to now, an experimental Ω format called Γ is in alpha stage now).

— ConTEXt supports by default a wide variety of graphics format, regardless of the output driver used. Direct inclusion of MetaPost-produced EPS graphics in PDF is readily available thanks to ConTEXt macros (these same macros can now be used in LaTeX.); for other output formats, figure dimensions for most common formats are extracted by the figure files themselves.

— ConTEXt support is provided directly by its author. Feature requests are usually implemented in a very brief time and added to the core on the subsequent ConTEXt release. Also, the ConTEXt sources (or at least the core modules) are usually duly documented, and can make an interesting (and instructive) reading since comments on why a particular hack was used are imbedded in the sources. Such sources can be compiled into colorful, readable documents by the `texexec.pl` script.

On the other hand, LaTeX has its solid advantages:

— LaTeX is modular. This means that a basic LaTeX can be used on a smaller, slower computer, with little resource consumption. ConTEXt’s format, on the other hand, has now reached five megabytes in size, and all this stuff has to be kept in memory even if the advanced features are not used.

— LaTeX is standard in the scientific fields. Technical documentation, and especially scientific articles and other contributions to scientific newspaper is usually expected to be in LaTeX. Moreover, $\mathcal{AMS}$ extensions make complex formulas easy to write. (ConTEXt is now trying to bridge this gap.)

— LaTeX is well documented. Having been around for a longer time, LaTeX has a larger user base and a wider range of documentation. ConTeXt documentation is scarce and is now getting outdated.

Overall, my suggestion is to use the appropriate tool for the needs; for technical stuff with no complex formatting requirements, LaTeX is probably the best solution. Nontechnical, creative, complex stuff is more easily managed in ConTeXt.