Comparison of OpTEX with other formats: LATEX and ConTEXt

Petr Olšák

Introduction

OpTEX [1] was introduced in an article [2] in the previous issue of *TUGboat*. It is a macro package that creates a format for LuaTEX. Its features are comparable with other formats like LATEX or ConTEXt. One may ask why to use a new format, particularly when it requires a different markup syntax. I try to answer this question here. I present a comparison among the LATEX, ConTEXt, and OpTEX formats, from various points of view.

Basic concept

LATEX. It was created in the 1980s as a real *format*, i.e. the implementation of visual and typographical aspects of the TEX output. Moreover, it provides a markup language given in [3] which was intended for the authors of (typical) scientific publications. Authors are instructed: use this markup and don't worry about the typographical look of the output. This look is implemented for you in the format.

A very important feature of LATEX is its modularity. There are macro packages that solve particular problems with the typesetting of documents which can be loaded when the document is processed. This concept has grown to a size which nobody could have expected in the 1980s. Now, some packages give an interface to authors to set various typographical parameters of the typesetting too. So, there is no single format of the output. But the possibility of controlling the visual aspect of the output has no uniform strategy. It is spread among various packages created by various authors.

LATEX introduces a new level of terminology, syntax, etc. over the TEX primitive level. We are not using *control sequences* (meaning macros, registers, ...) in LATEX. There are *commands* and newly *functions* and *variables* here. But the interpreter is TEX, so it reports (for example) the message "`undefined control sequence`" and the LATEX users may not know this term and they may not understand such messages. For example, a typical TEX message "`missing \cr inserted`" is not understandable for average LATEX users because they are using `\\`, not `\cr`.

The different LATEX syntax can be shown in the following example. The setting to the register which controls the width of the typesetting area is documented as

```
\setlength{\textwidth}{13cm}
```

which would be `\hsize=13cm` at the TEX primitive level. The primitive level is allowed in LATEX documents too, so we often see a mix of primitive syntax and LATEX syntax in real-world documents.

In the last ten years, a new language level over the primitive level has been developed, used, and propagated by LATEX team: expl3. It is intended to macro/package writers for LATEX. It is even further from the TEX primitive level. For example:

```
\tl_set:Nn \l_pkgname_hello_tl { Hello! }
```

is comparable to `\def\hello{Hello!}` from the primitive point of view. Very special naming conventions must be used here. And different terminology is used: the `\l_pkgname_hello_tl` is not a macro without parameters, but rather a *variable*. The `\tl_set:Nn` is not a macro that expands to the `\def` primitive, but rather a *function*.

ConTEXt. It was created and is still developed by Hans Hagen (and colleagues). The first released version was in 1994 (in Dutch) and the ConTEXt name was given to the package in 1996. Now, we have a development version 'ConTEXt lmtx' based on the LuaMetaTEX engine (not included in TEX Live) and the stable 'ConTEXt mkIV' based on LuaTEX engine. When I use the word ConTEXt in this article, I mean the stable ConTEXt, because I don't have experience with the development version.

ConTEXt is not only a format, it is a tool that enables to set the typesetting parameters consistently and process the document. All features used in typical present-day documents are supported in one place without the need to load external and third-party packages.

The settings of the typesetting parameters are done with `\setup...` commands in key-value syntax. So, new syntax over primitive TEX syntax is created here. And the distance from this level of syntax to the primitive level seems to be quite big. For example, the primitive `\hsize` is set when a parameter `width` in a special context is used in the ConTEXt.

ConTEXt is closely related with MetaPost for creating vector graphics which can be programmed and which "cooperates" with typesetting material. LATEX and OpTEX more commonly use Ti*k*Z for this, although all the formats support both graphics packages, among others.

OpTEX. It was released in 2020 and the first stable version dates from February 2021. It is the successor of the OPmac macros [4, 5] specially designed as a format for LuaTEX engine.

OpTEX is similar to plain TEX but provides myriad additional features needed when preparing typical documents in PDF format. The list of the features is presented in the next section.

OpTEX does not try to define a new level of language over the primitive level of TEX. It is intended that if something is not supported in OpTEX macros then the user works at the TEX primitive level (or plain TEX or OpTEX basic macro API). For example, when you want to set the width of the typesetting area, use simply `\hsize=13cm`.

The macros are straightforward, they solve only what is explicitly needed. They do not scan nor manipulate with lots of parameters for setting typography. OpTEX generates a "default typography" if nothing more is done. The main concept is: if you want different typography or different behavior, then copy appropriate macros from OpTEX kernel and do changes in them in your macro file. For example, suppose you want to give a different look to section titles. Then copy the macro `\_printsec` from OpTEX and modify it. You can see that the macro uses primitive commands for typesetting: `\hbox`, `\vbox`, `\kern`, `\vskip`, `\hskip`, `\penalty`, etc. You can use this box-penalty-glue concept directly without any inserted inter-layer of language. This is very natural in TEX and you can use the full power of TEX.

We can summarize the basic concept of OpTEX to two sentences: (1) We can return back to the TEX principles. (2) Simplicity is power.

Kernel versus packages

LATEX. The features implemented directly in the LATEX kernel (i.e. in the `.fmt` file) correspond to the time of LATEX's origin. There is no color support, no support for graphics insertions, no hyperlinks support, no Unicode fonts support, etc. All these additional features are solved using external packages. So, document preambles contain plenty of `\usepackage` commands to load additional macros. Without them, you cannot solve typical problems when processing today's documents.

ConTEXt. It has an almost monolithic kernel that implements all necessary features. There is a possibility of "modules" in ConTEXt (something comparable to LATEX packages) but they are not typically used because virtually all features are present in the kernel.

OpTEX. The kernel implements:

- all macros from plain TEX,
- font selection system for Unicode fonts,
- Unicode math,
- color support including color mixing,
- graphic insertions,
- creating simple graphics elements,
- typesetting at absolute or relative positions,
- external and internal references and hyperlinks,
- automatic generation of a table of contents,
- generation of `\cite` references from `.bib` files,
- creating alphabetically sorted indexes,
- hyphenation for all available languages,
- switching between language-dependent phrases,
- footnotes and marginal notes,
- verbatim listings including syntax highlighting,
- PDF outlines in Unicode,
- creating tables with a new `\table` macro,
- creating slides for presentations,
- simple predefined styles "letter" and "report",
- comfortable setting of page layout,
- printing "lorem ipsum dolor sit",
- simple API for macro writers,
- loops and key-value syntax for macro writers,
- namespaces for users and macro writers.

Many other features can be implemented in a small number of macro lines. They are listed in the OpTEX tricks web page [6]. Other such features will be added here if a user asks me to solve a new problem. A user can copy the macro lines from this web page to his/her macro file and (possibly) modify them and use them. Almost every feature listed here is typically comparable with using some LATEX package, but is solved with more straightforward macros. For example, the feature comparable with the `import` package is implemented by three lines of macro code in OpTEX trick 0035. The LATEX package `import` itself has 120 lines of macro code.

Some features take more than a few lines of code. OpTEX supports loading macro packages too, with a `\load` macro. For example, there are packages:

- `qrcode` calculates QR codes and prints them,
- `vlna` handles non-breakable spaces after Czech/Slovak prepositions and other similar typographical features, using the `luavlna` package
- `emoji` enables printing of a large number of colorized emoticons from the special Unicode font.

Documentation

LaTeX. The LaTeX project page [7] lists 8 files as "general documentation" with 160 pages in total, but this is not all. The mentioned documentation describes LaTeX kernel features. But users need to use dozens of packages when an average document is prepared. Each package has its documentation. This represents hundreds or thousands of additional documentation pages from various authors. The documentation is of different ages in different styles. It is difficult to recognize what is relevant and what is obsolete. There exists a book [8] that summarizes features of LaTeX and of all typically used LaTeX packages but not every LaTeX user has access to this book. And features of recent versions of packages may differ from those described in this book.

There is the LaTeX `doc` system: the macro programmer can write code and technical comments together. You need to pre-process these sources to get macro files usable when the format is generated or documents are processed. This system is widely used by LaTeX package programmers.

ConTeXt. There are about 180 PDF files with various ConTeXt documentation. It is not within the power of the average user to know such a huge amount of information and be able to select the most important parts when starting with ConTeXt. On the other hand, this illustrates that ConTeXt covers a very large area of computer typesetting.

OpTeX. The main OpTeX manual [9] is divided into two parts: user and technical documentation. The first one has only 22 pages. There is a summary of OpTeX markup at page 26. You can click on each control sequence listed here and the relevant part of user documentation is shown. You can click again on a control sequence here and your PDF viewer jumps to the second part with technical documentation and with a detailed technical description of the macro and with the macro source printed here.

The manual is created using OpTeX, of course. When the technical part of the documentation is processed, the actual OpTeX files with the macro sources and with detailed comments are read. It is similar to literate programming where technical notes and code sources are together in a common file. You don't have to pre-process this file: the files are ready to be read when OpTeX format is generated and when the documentation is prepared.

Of course, if the user wants information beyond document markup, then he or she must know the basics of TeX itself and plain TeX. This information is summarized in the document [10] at 26 pages. More information about TeX math typesetting and Unicode math is summarized in the document [11] at 30 pages.

The three documents mentioned above [9, 10 11] are sufficient to acquire all knowledge about OpTeX and TeX. They are all that you need to know when working with OpTeX documents. Of course, you can get more information about TeX engine used by OpTeX: [12, 13, 14]. No other documents are needed.

External programs

LaTeX. When creating the reference list of `\cite`d records from `.bib` file, LaTeX needs an external program (ancient BibTeX or newer Biber). Users have to run this sequence: `latex`, `biber`, `latex`, `latex` to get the correct reference list and `\cite` numbers.

When creating an index, the external program Makeindex or Xindy is used. The user must not forget to run `makeindex` followed by `latex` after final corrections (just before sending to print) are done.

When printing a code listing with syntax highlighting, some LaTeX packages use another external program, such as a python script.

OpTeX. We don't have to use any external programs when creating bibliography references, sorting an index, or printing code listings with syntax highlighting. All these tasks are done at the TeX macro level.

OpTeX reads the `.bib` file directly and creates the appropriate reference list. A bibliography style (a simple macro file) is used to set the rules for printing and sorting records. If all `\cite` commands are before the reference list generated by `\usebib`, then you need to run OpTeX only two times: the first run accumulates all labels used in `\cite` commands, and the records with these labels are read from the `.bib` file. The reference list is created and labels are connected to the generated numbers and saved to a `.ref` file. The second run uses these label-number pairs to print the numbers at the places where `\cite` commands are used.

Index sorting is done by a merge sort algorithm which is very efficient in the TeX macro expansion language. The special two-pass algorithm is used for similar phrases, which is configurable for almost all languages. The sorting rules are applied by the currently selected language.

The syntax highlighting of code listing is done at the TeX macro level. It is configured in macro files `hisyntax-c`, `hisyntax-html`, etc.

Font selection system

LaTeX. In the 1990s, the “New font selection scheme” (NFSS) was designed, along with the LaTeX2$_\varepsilon$ release. The system allows the selection of family, weight, shape, size, and encoding of the font independently. It was designed for old fonts with a maximum of 256 characters in the font. The NFSS creates a new layer over the `\font` primitive. Special internal names for font families were used and declared in font definition files (`.fd`). Each newly installed font was typically converted to various 8-bit encodings. The virtual fonts technique was frequently used. This required a high level of understanding fonts in TeX, making font installation almost impossible for average users.

When Unicode engines were released then a new `\fontspec` package for selecting Unicode fonts were designed. It follows the principles given by NFSS but adds new features to support Unicode text fonts.

OpTeX. The font selection system respects the basic plain TeX principle: you can select font variants typically by `\rm`, `\it`, `\bf` and `\bi` selectors when a font family is loaded by `\fontfam`. The variant is selected with respect to the current “font context” given by the current setting of font size and more features given by “font modifiers”. The set of font modifiers (for example `\cond`, `\caps`, `\sans`, `\bolder`) depends on features provided by the selected font family. Users can combine the font modifiers arbitrarily: the appropriate font is selected if the font family provides it. The font families including their modifiers are implemented in “font files”. The log file shows the available font modifiers of the selected font family. Users can create a font catalog with simply `\fontfam[catalog]\bye`. All families registered in font files are printed in this catalog; see [15]. You can see all available families, modifiers, and variants here. Font samples for each such combination are shown.

Unicode fonts are preferred in OpTeX. An appropriate Unicode math font is loaded automatically too when `\fontfam` is used. The special font modifier `\setff{...}` sets arbitrary OpenType font feature if it is supported by the font.

Of course, OpTeX supports usage of the `\font` primitive and allows you to simply incorporate the font selector (declared by `\font`) into your macros to enable scaling this font by the size context used in the document. You need not declare a `\font` for the same font repeatedly for all necessary sizes.

Fragile commands in titles

LaTeX. LaTeX users know the term “fragile command” well. A macro used in the title of section or chapter can be broken when it is written to internal files used for generating of the table of contents. It is been over 20 years that (ε-)TeX has provided the `\detokenize` and `\scantokens` primitives but LaTeX users are still fighting with fragile commands, solving the problem with methods like using `\protect` macro or `\DeclareRobustCommand` which adds a mysterious space to the end of the control sequence name, making tracing more difficult.

OpTeX. There are no “fragile macros”. OpTeX reads the titles for chapters and sections in verbatim mode and re-tokenizes them when they are actually used for printing. Moreover, the verbatim constructions work in titles too:

```
\sec Title with `\this{` matter
```

The in-line verbatim is surrounded by `` `...` `` here. As shown, unbalanced braces `{}` can be in the verbatim text. And this verbatim text is correctly printed in the table of content, headlines, and PDF outlines. This is difficult in LaTeX because the syntax for section parameters is the text surrounded by balanced braces `{}`. The title parameter in OpTeX is delimited by the end of the line.

ConTeXt. The title parameter is surrounded by braces `{}` similarly as in LaTeX, so the example above with `\this{` cannot work as-is in ConTeXt.

Markup language

LaTeX. Almost all user-level LaTeX macros have undelimited parameters, so users must use braces to specify parameters of more than one token. LaTeX’s `\newcommand` does not allow declaring a macro with delimited parameters. Brace pairs `{}` outside the context of macro parameters (i.e. in the meaning begingroup–endgroup) are not preferred in LaTeX markup language.

Writing the LaTeX document often means *coding* the document. There are many nested LaTeX environments (a new syntactic concept in LaTeX) and `{}` braces. For example, Beamer documents are more programming language than source text. The author’s ideas (what’s intended to be displayed in the Beamer presentation) disappear among the surrounding code in source files.

There is no “standard LaTeX markup language” in the sense that if a program (other than TeX) understands this markup then it knows how to convert from or to this markup. We cannot suppose what packages will be used, and they significantly affect the tagging of the document.

OpTEX. The main credo is: the source files of the document are created (typically) by humans, humans will read these sources and manipulate them. Source files are intended primarily for humans, not for machines. Machines must be programmed to respect these principles.

This is why OpTEX tries to define a more lightweight markup language. There are no highly nested environments, there is a minimal number of braces `{}`. Lists of items begin naturally with a `*` character, titles are terminated by the end of the line.

Many other relatively lightweight markup languages have been devised; Markdown, for example. But we cannot have absolute control of "to PDF processing" in Markdown. Maybe, an author can initially prepare documents in Markdown, but the result must be converted to a `.tex` format, which could use the OpTEX format. A (human) typesetter takes this `.tex` file and starts to program the typography of the document. It means that he or she manipulates with the `.tex` sources and adds more information here and adds `.tex` macro files.

The `.tex` sources are "the heart" of the document. They are what's needed to archive documents for future use (in the next edition, for example). They can be processed for printing purposes to PDF (by OpTEX) or they can be converted to other formats (by other converters).

There is an OpTEX markup language standard (OMLS) [16] which gives instructions for potential authors of converters from/to OpTEX format. The standard specifies the syntax and semantic of "known OpTEX tags". Other tags can be ignored. The OMLS covers tags that are used in typical tasks when processing OpTEX documents.

ConTEXt. The markup language is somewhere between LATEX coding and OpTEX writing the document. Tables, though, are created in a special way not typical in TEX. They look like HTML code which is far from a text intended for humans.

Hello world test

Create the following test file

```
\loggingall
\cslang              % Czech hyphenation patterns
\fontfam[Adventor]   % Unicode font family Adventor
Ahoj světe!          % Hello world!
\bye
```

and its counterparts in other two formats (LuaLATEX and ConTEXt), i.e. loading the Unicode Adventor family, selecting a language, and printing one sentence. We count the number of lines in the `.log` file when full tracing is activated by `\loggingall` and the time spend when processing this document without `\loggingall` (measured using the author's notebook). The following table summarizes the results:

	OpTEX	LuaLATEX	ConTEXt
lines	1.8 K	3.6 M	231 K
time	0.5 s	1.0 s	1.1 s

We can see that LuaLATEX needs to do about two thousand times more internal operations than OpTEX to process this "Hello world" document. LuaLATEX is slightly better than ConTEXt (although ConTEXt was run with `--once` option) in the time measurement, because ConTEXt has an exceptionally large `.fmt` file and time is spent loading and uncompressing this file. See the following table which shows the size of `.fmt` files in the GNU/Linux TEX Live distribution:

	OpTEX	LuaLATEX	ConTEXt
`.fmt` size	750 KB	1.2 MB	11 MB

Note that OpTEX has noticeably smaller format file than LuaLATEX although it implements not only the comparable features of LATEX kernel but also a number of LATEX packages not included in the kernel: `xcolor`, `hyperref`, `url`, `listings`, `biblatex`, `graphicx`, `geometry`, `amsmath`, `amsymb`, `fontspec`, `unicode-math`, `cprotect`, `eqparbox`, `tabularx`, `booktabs`, `keyval` and more.

Why it is possible? OpTEX keeps its basic concept "simplicity is power".

Moreover, OpTEX does not create any auxiliary file if it is not needed. When the example above was processed, LATEX creates an unnecessary `.aux` file and ConTEXt creates a `.tuc` file, but OpTEX creates only `.pdf` and `.log` files.

When OpTEX needs an auxiliary file, then only a `.ref` file is created where all needed information is saved. We don't need `.toc`, `.lot`, `.lof`, `.nav`, `.glo`, `.idx`, `.ind`, `.bbl` and others that have been used over the years.

Namespaces

LATEX. There is no user namespace. You cannot tell users: "you can define and use an arbitrary control sequence with given naming scheme". There is only the concept: "try it and maybe you will be successful". More precisely, the user can define an arbitrary control sequence using `\newcommand` and this macro ends with the error "control sequence defined already" (in other words: "you are out of your namespace"), or it is successfully defined.

LaTeX packages commonly use internal names containing the `@` character. These names typically begin with `pkg@` where `pkg` is the name of the package. The `expl3` language uses `_pkg_` in the names of control sequences too. This results in code where we repeatedly see `_pkg_`, `_pkg_`, `_pkg_` in practice, despite various language features to attempt to reduce the need for the repetition. This reduces the clarity of the code and reduces the concentration of an eventual reader on the key topic of the code.

OpTeX. We can say to users: Your namespace includes names with letters only. You can define control sequences in this namespace and use them. Moreover, some control sequences in your namespace have pre-declared meanings (primitive, macro from OpTeX, register or anything else). You can use them. If you don't know about the existence of the meaning of a pre-declared control sequence and you never use it with this meaning, then you can re-define it without problems. For example, `\def\fi{finito}` will work as you wish, if you never use `\fi` in its primitive meaning.

How does it work? All primitive control sequences and internal macros in OpTeX have their duplicates in the format `\_foo`. For example, there are control sequences `\hbox` and `\_hbox` with the same meaning. OpTeX macros use exclusively the `\_foo` form. This is the OpTeX namespace. Users can utilize `\_foo` sequences too without problems in "read-only" mode and they can re-define such sequences when they know what they are doing.*

The packages for OpTeX have their own namespace in the naming scheme `\_pkg_foo`. But the package writer doesn't have to write `_pkg_` repeatedly in the internal macros because there is a `\_namespace{pkg}` declaration. If it is used then the macro programmer can write `\.foo`, `\.bar` in the code, and it is transformed to `\_pkg_foo`, `\_pkg_bar` at the input processor level, when the macro file is scanned.

Happy (Op)TeXing!

* Unfortunately, there is one exception to this principle of the user namespace: the control sequence `\par` is hardwired in the TeX implementation. For example, it is generated at each empty line. The OpTeX manual mentions this exception from the user namespace. I wrote a patch to TeX, which enables to set any name to this hardwired control sequence. Unfortunately, I sent this patch after the deadline for TeX Live 2021, so we must wait a while until it will be implemented in the TeX engines. The patch is ready, documented, and tested on my computer. When the patch is applied then this footnote will be rendered obsolete, since there will be no exceptions from the user namespace.

References

1. OpTeX web pages. `petr.olsak.net/optex`
2. P. Olšák: OpTeX — A new generation of Plain TeX. *TUGboat* 41:3, 2020, pp. 348–354. `tug.org/TUGboat/tb41-3/tb129olsak-optex.pdf`
3. L. Lamport: *LaTeX: A document preparation system*. Reading, Mass.: Addison-Wesley, 1994.
4. OPmac web page. `petr.olsak.net/opmac-e.html`
5. P. Olšák: OPmac: Macros for Plain TeX. *TUGboat* 34:1, 2013, pp. 88–96. `tug.org/TUGboat/tb34-1/tb106olsak-opmac.pdf`
6. OpTeX tricks. `petr.olsak.net/optex/optex-tricks.html`
7. LaTeX Project web pages. `https://www.latex-project.org/`
8. F. Mittelbach et al.: *The LaTeX Companion*. Reading, Mass.: Addison-Wesley, 2004.
9. OpTeX manual. `petr.olsak.net/ftp/olsak/optex/optex-doc.pdf`
10. P. Olšák: TeX in a nutshell. 2020, 29 pp. `petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf` `ctan.org/pkg/tex-nutshell`
11. P. Olšák: Typesetting Math with OpTeX. `petr.olsak.net/ftp/olsak/optex/optex-math.pdf`
12. V. Eijkhout: *TeX by Topic*, 2007. `ctan.org/pkg/texbytopic`
13. D. Knuth: *The TeXbook*. Addison-Wesley, 1984.
14. LuaTeX reference manual. `www.pragma-ade.com/general/manuals/luatex.pdf`
15. Font catalog generated by OpTeX. `petr.olsak.net/ftp/olsak/optex/op-catalog.pdf`
16. OpTeX/ markup language standard (OMLS). `petr.olsak.net/ftp/olsak/optex/omls.pdf`

◇ Petr Olšák
Czech Technical University
in Prague
`https://petr.olsak.net`