OpTEX — A new generation of Plain TEX

Petr Olšák

Introduction

OpTEX [1] is a new format prepared for LuaTEX. It keeps Plain TEX's simplicity, adds the power of the OPmac macros [2] and leaves behind the old obscurities with non-Unicode fonts and with various TEX engines. It provides a powerful font selection system, colors, graphics, references, hyperlinks, syntax highlighting, preparing indexes and bibliography listings. All these features are implemented at TEX macro level and they are ready to use without any external program. OpTEX is a new Plain TEX suitable for the present.

OpTEX was introduced in February 2020 and uploaded to CTAN. Now, it is ready to use in both TEX Live and MiKTEX with the basic command `optex document`. It underwent significant development in the first half of 2020, so please take the most recent version of it (from CTAN or a distribution) if you want to experiment with it.

First example

A question was given on TeX.StackExchange [3]: "How can I write the symbol ţ in TEX?" Unfortunately, the accepted answer is typical for the old days of TEX: use a macro package which loads a font in an obscure 8-bit encoding and use the command `\textsubwedge{t}`. For me, the question has no sense and the answer sounds like something from the last millennium. Imagine a little modification of this question: "How can I write the symbol K in TEX?" And the answer should be: use package XY, then you can use `\printthischarK` command. But the normal answer is: "Use K in your text."

My answer, second on that StackExchange page, was: "Use OpTEX and then use normally the symbol ţ. If a Unicode font supporting this character is loaded* then there is no more problem." For example:

```
\fontfam[Linux Libertine]
Symbol ţ
\bye
```

There was an addendum to my answer, essentially this: You can define `\def\t{ţ}` if your text editor or keyboard does not comfortably support making the ţ. Of course, TEX supports the command `\def` for such situations (among many others).

Unfortunately, there are many similar "problems"; you can see them at StackExchange or elsewhere. These "problems" shouldn't exist if we leave the old way of thinking about TEX. Today, there are plenty of good Unicode fonts. Simply, use them.

The second typical matter can be found in the accepted answer of that same StackExchange thread: "If you are using a special TEX engine then you must do `\ifsomething ...\fi` and you must load a package XY supporting this `\ifsomething`." This is absurd. OpTEX eliminates the need to deal with such issues. It supports only one modern TEX engine: LuaTEX. Simplicity is power.

The three-line source file from the previous example shows another very important characteristic of Plain TEX. You need not *code* your document,** you don't need to adapt yourself to a special computer language for your source files where are many `\begin{foo}...\end{foo}`, for example. Simply *write* what you want. Of course, you have to decide what font family is used (the `\fontfam` command) and then just write the text. And use `\bye` if you want to say goodbye to TEX.

Main principles of OpTEX

There are three main principles.

- The first one was mentioned in the previous paragraph. The author can *write* the document, he or she need not *code* the document in a computer language. The source text of the OpTEX document keeps the basic rules about tagging documents (chapters, sections, emphasized text, footnotes, listings of items, etc.), but there are minimal tagging marks, because we want to keep the text human-readable. There is a minimum of braces `{...}`, for example.
- The second principle of OpTEX says: don't hide TEX. Don't declare new parameters and new syntax constructions. If a user needs or wants to use TEX then he or she simply uses TEX. For example, we have the `\hsize` primitive register, and we don't declare a new one like `\textwidth`. If you need to set a value to the register then use simply `\hsize=15cm`; there is no alternative syntax like `\setlength{\textwidth}{15cm}`. Basic knowledge of the TEX primitive syntax is expected when using OpTEX. But it pays off.

* More precisely: it is not a single character in this case but this is only a technical detail.

** Many questions at StackExchange begin with "My *code* is ...".

- LaTeX, ConTeXt and their many packages give users plenty of new parameters and options, and they create a new level of language (on top of TeX). When we are using or developing OpTeX, we don't need to go that same way. There is TeX for controlling the document, without new options and parameters. The macros of OpTeX are more straightforward and simple because they do not create a new level of syntax but only what is explicitly needed. If you need to make a change in the design of the document (for example) then you can copy the appropriate macro from OpTeX to your macro file and make the changes directly there. For example, there are macros `\_printchap`, `\_printsec` in OpTeX. Do you need a different design? Copy such macros to your macro file and declare your design there. This is the third principle of OpTeX which establishes a significant difference from LaTeX or ConTeXt and which keeps the macros simple.

Summary of features provided by OpTeX

The user manual of OpTeX is 21 pages. It contains hundreds of hyperlinks to a second part of the manual: technical documentation (about 140 pages). The technical documentation is generated directly from the OpTeX sources. There are listings of all OpTeX codes with extensive technical notes about the code.

We introduce a few features of the OpTeX system here, giving just short overviews.

The font selection system. You can use basic *variant selectors* `\rm`, `\bf`, `\it` and `\bi` as in Plain TeX, i.e. `{\bf text}`. Plain TeX does not define the `\bi` selector for bold italic, but OpTeX does because almost all font families used today provide this font variant.

You can choose the font family by the `\fontfam` command. WYSIWYG systems typically offer a menu for selecting the font family and this menu shows how text looks in listed fonts. This is a great advantage of such systems. You can get similar information by writing `\fontfam[catalog]`. Then all font families registered with the OpTeX macros are printed like a font catalog. Each font family is shown in all provided variants and the font modifiers given for each family are listed too. This "almost instantaneous" font catalog provides a sort of substitute for the interactive menus used in WYSIWYG systems.

Many font families provide *font modifiers*, for example `\cond` for condensed variants or `\caps` for capitals and small capitals. Usage of such font modifiers change a *font context*, but does not select the new font directly. This is done only when a variant selector is used. The variant selector respects the font context given by previous font modifiers. For example `\cond\it` selects condensed italics and if someone uses `\bf` in the same TeX group scope where `\cond` was declared then the bold condensed variant is selected.

There are many font modifiers declared among the font families. The set of available font modifiers depends on the selected font family. These modifiers can be independent of each other if the font family provides all such shapes. For example, `\cond` and `\caps` are independent, so you can set four font contexts by these two selectors and you can use four basic variant selectors: this gives 16 font shapes.

The settings of Unicode font features are implemented as font modifiers. This means that the current setting of font features is a part of the font context.

The setting of the font size is implemented as another font modifier. It means that the font size is the part of the font context too. If the family provides optical sizes, these sizes are respected by the OpTeX font selection system.

The font families are declared and registered by the font selection system in *font family files*. The family-dependent font modifiers are declared here. You can load more font families (by more `\fontfam` commands) and you can select between them by the *family selectors* like `\Termes`, `\Heros`, `\LMfonts`, etc. The font modifiers and variant selectors behave independently in each family, and they respect the selected family. For example if you want to mix Heros with Termes, you can declare:

```
\fontfam[Heros]
\fontfam[Termes] % Termes is current
\def\exhcorr{\setfontsize{mag.88}}
\famvardef\ss {\Heros\exhcorr\rm}
\famvardef\ssi{\Heros\exhcorr\it}

Compare ex-height of Termes
\ss with Heros \rm and Termes again.
```

This example shows several things:

- If multiple font families are loaded then the last one is selected as the current family.
- More variant selectors can be defined by the `\famvardef` command. The example shows a declaration of new `\ss` (sans serif) and `\ssi` (sans serif italic) variant selectors.

- The font size can be set by the `\setfontsize` command. It provides more syntactic rules but one of them is the keyword `mag`: the new font size is calculated as a factor of the font size currently selected.
- The Termes and Heros families have visually incompatible x-heights. We need to do the correction Termes = 0.88 Heros.

Macro programmers can declare *font selectors* directly with the `\font` primitive if the font name or font file name and its font features is known. Or the font selector can be declared by the `\fontlet\new=\ori` ⟨*sizespec*⟩ if another font selector is known, and we need only to set another font size of the same font. Finally, the font selector can be declared by the `\fontdef` macro if you can set the font by variant selector and font modifiers.

The last case (by the `\fontdef` macro) respects the actual font family and font context when the `\fontdef` macro is used. If you change the font family before a set of `\fontdef` declarations then all declarations are re-calculated to the new font family. Another example: you can set a new default font size by `\typosize[11/13.5]` and all fonts declared by `\fontdef` will respect this new font size.

OpTEX loads only a few 8-bit Latin Modern fonts when its format is initialized. The Unicode fonts cannot be here due to a technical limitation of LuaTEX. It is supposed that these pre-loaded fonts will be used only for short experiments with OpTEX macros, not for processing real documents. A user should specify the font family with `\fontfam` first. This macro loads the Unicode variant of the fonts.

A lot of font families provided by OpTEX have registered the appropriate Unicode math font too. For example Latin Modern fonts have Latin Modern Math, Termes has TEX Gyre Termes Math, etc. The `\fontfam` macro loads this Unicode math font too unless the user says `\noloadmath`.

Tagging the document. All the typical tags for documents are borrowed from OPmac. The `op-demo.tex` document shows the basics of such tagging. Chapters are marked by `\chap` ⟨*title*⟩, sections by `\sec` ⟨*title*⟩ and subsections by `\secc` ⟨*title*⟩. The ⟨*title*⟩ ends at the end of the current line (unless the line ends with the `%` character; then the title continues). This decision mainly respects user needs: to *write* the document simply. Today, long lines (more than 80 characters) are quite common. Macro writers have a little complication if they use `\chap`, etc., in their macros because the end of line is changed locally only at the input processor level, but this can be handled.

Labels for cross-references can be declared by `\label[`⟨*label*⟩`]` or by `\sec[`⟨*label*⟩`]` ⟨*title*⟩, etc. The labels can be used by the commands `\ref[`⟨*label*⟩`]` or `\pgref[`⟨*label*⟩`]`.

Lists of items look like:

```
\begitems
* First idea.
* Second idea.
  \begitems \style i
  * First subidea.
  * Second subidea.
  \enditems
\enditems
```

Auto-generated listings. The table of contents can be printed by the `\maketoc` command and the index by the `\makeindex` command. The alphabetical sorting of the index is done at the TEX macro level with respect to the rules of the current language selected. No external software is needed. Thus, you don't need to do more than specify the words to be indexed and write `\makeindex`. The index is reinitialized in each TEX run.

The situation is similar with bibliographies. OpTEX reads `.bib` database files directly without the need of any external program and creates these listings with respect to a big set of rules declared in the *bib-style files*. These rules and customizing possibilities are described in [4]. OpTEX provides `simple` and `iso690` bib-style files now.

Colors. Colors can be simply used, for example, `{\Red something}` or `{\Magenta text}`. They can be defined in three possible ways:

```
\def \Red     {\setrgbcolor  {1 0 0}}
\def \Magenta {\setcmykcolor {0 1 0 0}}
\def \Brown   {\setcmykcolor {0 0.67 0.67 0.5}}
\def \Black   {\setgreycolor {0}}
```

and the user can define more such colors. The respective color model (RGB or CMYK) is used in low-level PDF commands. If you need not mix color models in the PDF output then you can say `\onlyrgb` or `\onlycmyk`. Then the colors are recalculated to the desired color model as needed. This calculation is done only via simple math formulae; the visual feeling of the color may be changed. These two color models are not transformable from one to the other without loss of information.

OpTEX initializes two color stacks: one for normal text and a second for footnotes. Footnotes can span from one page to another independently on the main text, so we need two independent color stacks. You can create a long footnote in green, for example, with the main text in red at the same point.

The next page continues with the red main text and green footnote. All colors work without problems on the next page. (If you try to do the same in LaTeX, you realize that it does not work without special care.)

The color blender macro `\colordef` is provided by OpTeX. It enables color mixing in the subtractive (CMYK) or additive (RGB) color model.

Graphics. The `\inspic {`⟨*file name*⟩`}` includes a graphics file in JPEG or PNG or PDF format (the last can be a vector graphic) at the current typesetting point as an `\hbox`. The width or height of the picture can be given by `\picwidth` or `\picheight` parameters. Other parameters accepted by the `\pdfximage` primitive can be specified too. For example, you can select a given page from the included PDF file.

Inkscape (a free vector graphics editor) is able to save a vector graphic to a PDF file and labels to a LaTeX file. OpTeX is able to read both these files (the LaTeX commands used by Inkscape must be emulated here). You can do this by `\inkinspic` macro which outputs the PDF graphic plus the labels. They are printed in the current fonts selected in the document.

OpTeX supports linear transformations using commands `\pdfrotate`, `\pdfscale` and (in general) `\pdfsetmatrix`. All compositions of these operations are allowed too. The `\transformbox` macro does linear transformations and the real boundaries of the box are calculated in respect of the transformed material.

If the graphics need to interact with the text, then Ti*k*Z can be used (`\input tikz`). This works in Plain TeX too. But simple tasks can be done using OpTeX macros without Ti*k*Z (we are happy when Ti*k*Z is not loaded because Ti*k*Z is a very big package). For example, putting the text into an oval or into an ellipse (its size depends on the amount of the text) can be done directly by `\inoval` or `\incircle` OpTeX macros. A clipping path can be declared by `\clipinoval` or `\clipincircle`.

Hyperlinks. There are four types of internal links: cross-references, citations (bibliography), links from the table of contents or index, and hyperlinks to/from footnotes. There is one type of external link generated by `\url` or `\ulink` macros. The hyperlinks can be activated by the `\hyperlinks` or `\fnotelinks` commands. The user or macro programmer can declare more types of hyperlinks.

Structured outlines (for PDF viewers) are automatically generated by the `\outlines` macro.

Verbatim text. Code listings can be placed between a `\begtt` and `\endtt` pair, or they can be included from an external file with, e.g., `\verbinput (`⟨*fromline*⟩`-`⟨*toline*⟩`) filename.c`. Inline verbatim text can be surrounded by an arbitrary character declared by the `\activettchar` macro. Nowadays, the most common usage is `\activettchar‘` as a declaration. Then you can type `‘\relax‘` to print `\relax`. This tagging is inspired by the Markdown language and is used very commonly at StackExchange, for example.

Sometimes you need to use inline verbatim in titles or parameters of other macros. This doesn't work when the `\activettchar` character is used because there is a "catcode movement" in the parameter of `‘...‘`. OpTeX provides a robust alternative command for such situations: `\code{`⟨*text*⟩`}`. The ⟨*text*⟩ is printed detokenized with `\escapechar` set to −1. From the user point of view, all "sensitive" characters in the `\code` parameter ⟨*text*⟩ should be escaped. For example, `\code{\\relax\{}` prints `\relax{`. This can be used in titles of sections, etc., without problems.

Listings can be printed with highlighted syntax (typically colored). Such syntax highlighting is defined in *hisyntax* macro files and can be activated with `\begtt \hisyntax{C} ... \endtt`, for example. All processing is done at the TeX macro level without using any external programs. These *hisyntax* files are easily customizable. They support C, XML/HTML, TeX and Python syntax at this time; others may be added in the future. Users can declare more such files.

Languages. LuaTeX is the only TeX engine which enables loading hyphenation patterns for a selected language on demand inside the document. Thus, we need not preload all hyphenation patterns in the format. Hooray! OpTeX provides the language selectors `\`⟨*isocode*⟩`lang` (for example `\enlang`, `\frlang`, `\delang`, `\eslang`, `\cslang`). These commands load the hyphenation patterns of a given language when they are first used in the document, and switch to the loaded hyphenation patterns when they are used subsequently. Macro programmers can set more language-dependent macros; these macros are processed when an `\`⟨*isocode*⟩`lang` language selector is used.

Language-dependent phrases like "Chapter", "Figure", "Table" are automatically selected by the current value of the `\language` primitive register (this is used for hyphenation patterns). These phrases are declared in OpTeX via:

```
\_langw en  Chapter   Table    Figure    Subject
%-----------------------------------------------
\_langw cs  Kapitola  Tabulka  Obrázek   Věc
\_langw de  Kapitel   Tabelle  Abbildung Betreff
\_langw es  Capítulo  Tabla    Figura    Sujeto
```

Quotation mark pairs can be declared by `\quoteschars`⟨*clqq*⟩⟨*crqq*⟩⟨*clq*⟩⟨*crq*⟩, for example `\quoteschars“”‘’` for English. The first type of quotation marks can be printed by `\"text"` and the second type by `\'text'`.* Several languages have their `\quoteschars` predefined in OpTEX.

Styles in OpTEX. The default design style of the document is inspired by Plain TEX: 10 pt/12 pt size of basic text, 20 pt `\parindent`, zero `\parskip`.

The command `\report` at the beginning of the document sets some typesetting parameters differently, suitable for reports. The `\letter` command sets a design convenient for letters.

If you write `\slides` then you can create presentation slides. This style is documented in the file `op-slides.tex` which also serves as an example of usage of this style.

Name spaces for control sequences. Suppose that the user writes `\def\fi{Finito}` into the document. What happens? When LATEX, ConTEXt or original Plain TEX is used then the document processing crashes. When OpTEX is used, then nothing critical happens. The user name space of control sequences allows names where only letters are used. If such sequences are redefined by users then this only affects their own usage and macros; it's not a problem for the internal macros of OpTEX. The internal macros of OpTEX do not use such control sequences.**

When OpTEX initializes, all TEX primitives and OpTEX macros have two representations, prefixed: `\_hbox` and unprefixed: `\hbox`. OpTEX uses only the prefixed versions. This is the OpTEX name space. A user can work with the non-prefixed versions of control sequences. If he or she redefines them nothing happens with the OpTEX internal macros.

The internal OpTEX macros not intended for direct usage by the user have only a prefixed form. And the control sequences never used in the OpTEX macros but offered to the user (`\alpha` and other sequences for math symbols) are defined only in unprefixed form (in the user name space).

The character `_` always has category code 11 (letter) in OpTEX. You aren't forced to write `\makeatletter` or anything similar when you need to access the control sequences from the OpTEX name space. Simply use them. You can redefine these control sequences, but then presumably you know what you are doing. An example of when it's expected to redefine macros from the OpTEX name space was given above where the macros `\_printchap` and `\_printsec` were mentioned.

The character `_` has category code 11 in math mode too. It is defined as math-active for doing subscripts in math formulae. Cases like `\int_a^b` work too because they are handled in the LuaTEX input processor.

OpTEX uses the `_` character only as the first character of control sequences. We suppose that macro package writers will use internal control sequences in the form `\_pkg_foo`. This is a package name space. Moreover, the macro writer does not need to see repeated `\_pkg_foo`, `\_pkg_bar`, `\_pkg_other` control sequences in the code because there is the command `\_namespace{pkg}`. When it is used then the macro writer can use `\.foo`, `\.bar`, `\.other` in the code which is much more human-readable. These control sequences are converted to internal `\_pkg_foo`, etc., automatically by the LuaTEX input processor.

Odds and ends. Logos are defined with an optional `/` character which can follow the control sequence; it is ignored if present. You can write, for example:

```
\OpTeX/ is a new generation of Plain \TeX/
with features comparable to \LaTeX.
But \LaTeX/ needs to load about ten additional
packages to have comparable features.
```

This source looks more attractive. We needn't separate such control sequences by `{}` or some similar construction.

The command `\lorem[`⟨*from*⟩`-`⟨*to*⟩`]` produces the text “Lorem ipsum dolor sit”. There is an interesting implementation of this macro: the 150 paragraphs of the text are not loaded into the OpTEX format. Rather, the first usage of the `\lorem` command loads the external file `lipsum.ltd.tex` from

* When `\quoteschars` are declared, then the original Plain TEX macros `\"` and `\'` are redefined. This problem is discussed further in the following section about compatibility with Plain TEX.

** There is only one exception: the control sequence `\par` is (unfortunately) hardwired to the TEX internal algorithms — it is the output of the tokenizer when an empty line occurs in the input. If the user redefines `\par` by mistake then processing may crash.

the LaTeX package `lipsum` and prints the given paragraphs. The second and subsequent usage of the `\lorem` macro prints the desired text from memory.

Compatibility with Plain TeX. All Plain TeX macros were re-implemented in OpTeX; nearly all features of Plain TeX are essentially preserved. But there are some differences.

- The internal control sequences like `\p@` or `\f@@t` were renamed or completely removed. We don't support the "catcode dancing" with the `@` character.
- The Latin Modern 8-bit fonts in the EC encoding are preloaded instead of the Computer Modern 7-bit fonts.
- The math fonts are preloaded in 7-bit versions comparable to Plain TeX plus AMSTeX. But if the `\fontfam` command is used then the preloaded 8-bit text fonts and 7-bit math fonts are not used; instead, Unicode text and Unicode math fonts are used.
- The control sequences for characters defined by `\mathhexbox` or in a similarly obscure way are not defined (for example `\P`, `\L`). We suppose that such characters should be used directly in Unicode. Only the `\copyright` macro is kept but it is defined by `\def\copyright{©}`.
- The accent macros `\"`, `\'`, `\v`, `\u`, `\=`, `\^`, `\.`, `\H`, `\~`, `\‘`, `\t` are undefined in OpTeX. We are using Unicode, so all accented characters can be written directly and this is the only recommended way. You can use these control sequences for your own purposes. For example `\"` and `\'` are used for quotation marks when `\quoteschars` are declared, as mentioned earlier. If you insist on using old accents from Plain TeX then you can use the `\oldaccents` command.
- The default paper size is A4 with 2.5 cm margins, not letter with 1 in margins. You can declare the default Plain TeX margins by the command `\margins/1 letter (1,1,1,1)in`.
- The page origin is at the top left page corner, not at the coordinates [1 in, 1 in] as in Plain TeX. This is a much more natural setting. These "1 in" values brought only unnecessary complications for macro programmers.
- The `\sec` macro is reserved for sections, not the math secant operator.

Tips and tricks. The web page [5] has a section Tips and Tricks when using OpTeX. This is inspired by Tips and Tricks of OPmac [6]. OpTeX users can give a problem and I'll try to put the solution here.

My path from TeX to OpTeX

My first attempts at TeX were with Plain TeX in 1991. I realized that it was unusable with the Czech language until 8-bit support of fonts was ready. The concept of writing `Ol\v s\'ak` instead of directly `Olšák` is not viable for real Czech texts. And Czech hyphenation patterns cannot work in the former case either.

I became a member of the development team of CSTeX. The 8-bit fonts with Czech and Slovak alphabet (CSfonts) were created using METAFONT, derived from the Computer Modern fonts. I created a macro to read the `plain.tex` file without the part of loading Computer Modern fonts. This part was replaced by loading CSfonts. The Czech and Slovak hyphenation patters were added and the CSplain format was originated. I was (and still am) a maintainer of CSplain. In the mid-1990s, I added PostScript support to CSfonts and Czech and Slovak accents for the 35 base Adobe fonts.

I read *The TeXbook* and felt that the description of all TeX algorithms could be done more systematically and clearly. This was the reason why I wrote the *TeXbook naruby* ("TeXbook inside out", in Czech only) [7]. This book became a standard for TeX manuals in Czech. For example, computer science students were using it to help welcome their new colleagues.

LaTeX was not the center of my interest because it seems to be more complicated. I wrote an article "Why I don't like using LaTeX" (1997, in Czech) [8]. I have reread this article recently and I found that all its arguments are still valid. The main problem is that LaTeX offers a "coding language" not a "human language" for writing documents. The second problem is that it tries to hide TeX by creating a new level of language. But this is impossible because TeX was not designed for such a task. LaTeX users will still see the TeX messages like "extra alignment tab has been changed to `\cr`". This language is different from the language used in LaTeX manuals, so LaTeX users are lost. From my point of view, it is unfair to TeX users to hide TeX.

I created encTeX in 2003 [9]. It is a pdfTeX extension which supports input of UTF-8 encoded documents directly. The Unicode characters (represented by multi-byte sequences in UTF-8) are mapped by encTeX to one 8-bit character or to a control sequence which can be defined arbitrarily. The most important advantage is that each Unicode character is represented as only one token in TeX. This is the main difference from the LaTeX package `\inputenc`.

I have been giving lessons about TEX to our students and learning from them. They represent a new generation and their point of view and requirements to TEX are very important to me. One of the results of these discussions is: at present, only Unicode makes sense. I decided that encTEX development was a dead end. We have a sufficient number of quality Unicode fonts today, we don't need to do special mappings and complicated macros, we can use Unicode fonts directly.

As a maintainer of CSplain and (of course) user of Plain TEX, I created many typical macros needed for document processing: creating the table of contents, fonts in different sizes, references, hyperlinks, etc. I released these home-made macros in 2013 as an additional package in CSplain called OPmac [2]. It works with CSplain or Plain TEX with all typical TEX engines. The great disadvantage of OPmac is that its technical documentation, though extensive, is only in the Czech language, because it was created as home-made documentation of home-made macros.

I created the template for student theses at our university based on OPmac and CSplain [10]. There are hundreds of satisfied users. It shows that Plain TEX is still viable today.

I planned to make a re-implementation of OPmac with new English documentation and with new features and internals. What TEX engine would be suitable for such a plan? XETEX does not support all of the micro-typographic extensions introduced by pdfTEX, and does not offer the extensive customization of LuaTEX. Furthermore, it seems that XETEX is no longer significantly developed, while LuaTEX has been declared substantively stable as of version 1.10 (`http://www.luatex.org/roadmap.html`). LuaTEX won.

I finished my reimplementation of OPmac by May 2020 and the result is called OpTEX.* The documentation of OpTEX expects knowledge of TEX basics. This is the main reason why I wrote a short encyclopedic document "TEX in a Nutshell" [11] (in English). The big reference "TEXbook naruby" [7] and short summary "TEX pro pragmatiky" [12] are available only in the Czech language, unfortunately.

I hope that OpTEX will find many users and thus gain more respect. I hope that it will be a good alternative to other currently used formats. It can show that the native principles of TEX do not have to be covered by new levels of computer languages, and they can live at present: 40 years after the birth of TEX. My dream is to eliminate the widespread notion among TEX users that TEX is equal to LATEX. Will you help me with it?

* You can guess why I had more time to do it in the year 2020.

References

1. `http://petr.olsak.net/optex`
2. P. Olšák: OPmac: Macros for Plain TEX. *TUGboat* 34:1, 2013, pp. 88–96. `petr.olsak.net/opmac-e.html` `tug.org/TUGboat/tb34-1/tb106olsak-opmac.pdf`
3. `tex.stackexchange.com/questions/541064`
4. P. Olšák: OPmac-bib: Citations using `*.bib` files with no external program. *TUGboat* 37:1, 2016, pp. 71–78. `tug.org/TUGboat/tb37-1/tb115olsak-bib.pdf`
5. `petr.olsak.net/optex/optex-tricks.html`
6. `petr.olsak.net/opmac-tricks-e.html`
7. P. Olšák: *TEXbook naruby*, 1996, 2000. 468 pp., ISBN 80-7302-007-6. Freely available. `petr.olsak.net/tbn.html`
8. P. Olšák: *Proč nerad používám LATEX*, 1997. `petr.olsak.net/ftp/olsak/bulletin/nolatex.pdf`
9. `petr.olsak.net/enctex.html` (2003–)
10. P. Olšák: The CTUstyle template for student theses. *TUGboat* 36:2, 2015, pp. 130–132. `petr.olsak.net/ctustyle.html` `tug.org/TUGboat/tb36-2/tb113olsak.pdf`
11. `ctan.org/pkg/tex-nutshell` (2020–)
12. P. Olšák: *TEX pro pragmatiky*, 2013, 2016. 148 pp, ISBN 978-80-901950-1-1. Freely available. `petr.olsak.net/tpp.html`

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