PanOpTEXum

Pandoc[1] Writer for OpTEX[2]

Version 0.1.0

Jan Martínek

[1] A universal document converter https://www.pandoc.org/
[2] LuaTeX format with extended plainTeX macros https://petr.olsak.net/optex/

Table of contents

Chapter 1
What is PanOpTEXum?

PanOpTEXum is a project that implements the OpTEX format into Pandoc. Many OpTEX users have expressed their desire to be able to convert documents to and from OpTEX using Pandoc. Currently only the writer is implemented, that means that you can use Pandoc to convert documents to OpTEX, but not from OpTEX as of yet.

1.1 Why does it exist?

This project was created as a semestral project at FIT CTU in Prague in the academic year 2023/2024.

Chapter 2
User documentation

2.1 Prerequisites

You need just one thing (or possibly two) before you should attempt to use or install this project:

- Working pandoc instalation.
  - That is easy to verify, just type:

    `pandoc -v`
  - If the above command prints anything other than pandoc's version, it means you do not have pandoc on `PATH` or installed.
- Working optex installation.
  - **Only if you want to convert the resulting files to pdf.**
  - This is also rather easy to verify:

    `optex -v`

2.2 Installation

Pandoc has the option `--data-dir`, by default it uses `~/.local/share/pandoc/`. Running `make install` will prepare files into the data directory. After that you can start converting to OpTEX as specified in the Usage section.

2.3 Usage

Simply use pandoc as you normaly would, however this time specify the `-d/--defaults` option as such:

```
pandoc -d optex ...
```

For example, to create the `.tex` file, that is used to create the `.pdf` of this manual, use

```
pandoc -d optex README.md >README.tex
```

After which you can use

```
optex README
```

to create `README.pdf` (you might need to run the optex command above 2-3 times, this is due to how optex handles table of contents etc.).

2.3.1 Options

Here is a list of all options that can be specified using pandoc's metadata fields.

2.3.1.1 Title

`title`

2.3.1.2 Subtitle

`subtitle`

2.3.1.3 Version

`version`

2.3.1.4 Author

`author`

2.3.1.5 Style

OpTEX supports three predefined styles:

1. report
2. letter
3. slides

You can specify which one you want to use, by setting the metadata field `style` to one of them. Alternatively, you can leave the `style` field unset, in which case it will behave just like OpTEX by default.

2.3.1.6 Table of contents

If you want to generate table of contents, please specify the metadata option `toc` as `true`. Use `tocname` to create a name for the **Table of contents** section.

Examples
Telling OpTEX to generate a table of contents:

```
toc: true
```

or

```
pandoc ... -M toc=true ...
```

Setting the name of the toc section:

```
toc: true # needs to be true, otherwise tocname hase no effect
tocname: Table of contents
```

or

```
pandoc ... -M toc=true -M tocname="Table of contents" ...
```

2.4 Complications

According to the Pandoc documentation, the option:

```
-t FORMAT, -w FORMAT, --to=FORMAT, --write=FORMAT
```

can also specify:

- the path of a custom Lua writer, see Custom readers and writers below

In the very same documentation we can also learn that, when writing a Defaults file, you can use environment variables:

> In fields that expect a file path (or list of file paths), the following syntax may be used to interpolate environment variables:
>
> ```
> csl:  ${HOME}/mycsldir/special.csl
> ```
>
> $USERDATA may also be used; this will always resolve to the user data directory that is current when the defaults file is parsed, regardless of the setting of the environment variable USERDATA.

One might suppose that when specifying the field `write`, which accepts a path to a custom Lua writer, it ought to be possible to use environment variables and expect them to get expanded. Alas, this is not true, which means that the install script needs to use the absolute path. This means that if the users data directory gets changed, the defaults file will **not** reflect these changes.

I am currently planning to create an issue in the Pandoc repository, because I believe that this behaviour directly contradicts the provided documentation.

Update: So there is an issue here that seems to talk about this problem, however it does not seem to have gained much traction…

Chapter 3
Ideas for the future

While I will probably add some of these features in the future, as this is not paid work, it will have to wait for when I have free time to do so.

3.1 Arbitrary heading levels

Currently we support heading levels only up to 4, however many formats might have the limit much higher or have no limit at all. Thus it would be prudent to support converting from these languages as well, maybe by creating an OpTEX macro, that would take 2 arguments:

1. The content of the heading.
2. The level of the heading.

The macro could then look like so:

```
\def\heading #1 #2 {%
%the definition of the macro would be here
}
```

This macro would essentially do one of three things:

1. Use an existing heading level if it already exists in OpTEX out of the box.
2. Use an already defined custom heading level, define by a previous occurence of this macro.
3. If none of the above are possible, define a new heading level and use it.

There might not be a need for this macro after all, as OpTEX provides the macro `\secl` .

3.2 Figure captions

Currently we are skipping any caption information. OpTEX supports this so the writer should as well.

3.3 Columns/rows spanning multiple cells in a table

This is also currently not supported by the writer, though OpTEX supports it.

3.4 Table captions

Same as **Figure captions**

3.5 Both bold and italic text at the same time

- **Bold**
- *Italic*
- ***Bitalic***

This issue has been rectified, by using a small trick. The chapter 1.3.1 of the OpTEX documentation describes the macro `\em` , which is perfect for our usage here. Thus this feature has been implemented and the transformations are as such:

Example input	Writer output	How it looks
`**bold**`	`{bf bold}`	**bold**
`*italic*`	`{em italic}`	*italic*
`***bold italic***`	`{bf {em bold italic}}`	***bold italic***

3.6 Font selection

Add the possibility to select a font.

3.7 Margins and paper size

Chapter 4
Testing the writer

4.1 Writing tests

Testing is implemented via the amazing bats framework. Currently, we are using it as a dependency via *git submodules*.

4.2 Running the tests

In order to run the tests, simply type:

```
make test
```

The way to run the tests (meaning the command used to execute the test runner) may be changed in the future from `make`, as I have my doubts regarding the usability of `make` across different platforms, mainly those that aren't unix-like, such as MS Windows.

Chapter 5
Technical documentation

5.1 Writer

The writer is written in the Lua language.

```
Writer = pandoc.scaffolding.Writer
```

optex-writer.lua

5.2 Template

The template is written in TEX using OpTEX macros and special pandoc markers.
optex-template.tex

5.3 Defaults file

There are now two defaults files:

- one for producing standalone versions of OpTEX documents, ready for TEXing,
  - it is this one: optex-default-standalone.yaml
- and another one for creating non-standalone versions, suitable for beign embedded into other OpTEX documents.
  - that is this one: optex-default.yaml

The defaults files are written in the `yaml` format as specified by the Pandoc documentation. For example, here is what it might look like:

```
standalone: true
template: ${USERDATA}/templates/optex-template.tex
pdf-engine: lualatex
```