Getting started with plasTeX

December 30, 2022

1 Installation

You will need a decent python to use plasTeX, at least python 3.6 (note that python versions older than 3.7 are no longer supported by the python software foundation). You can install plasTeX like any other python package, using `pip install plasTeX` (or `pip3` or `pipx` depending on your setup).

If you want a cutting-edge version, you can download the most recent code using git with

```
git clone https://github.com/plastex/plastex.git
```

(or by hand at this url) and then install it by running `python3 setup.py` in the plastex folder.

2 HTML rendering

2.1 Basic rendering

While plasTeX is a general purpose LaTeX document processing framework, its most common use is to convert LaTeX documents to HTML web documents. If your document `my_doc.tex` does not use any complicated LaTeX package, you can simply convert it by running `plastex my_doc.tex`

This will create a folder `my_doc` containing an HTML version of your document.

There are many command line options you can use to configure the way your document is rendered. The most important one is probably `--split-level=N` which specifies the highest section level that generates a new file. Each section in a LaTeX document has a number associated with its hierarchical level. These levels are -2 for the document, -1 for parts, 0 for chapters, 1 for sections, 2 for subsections, 3 for subsubsections, 4 for paragraphs, and 5 for subparagraphs. A new file will be generated for every section in the hierarchy with a value less than or equal to the value of this option. This means that for the default value of 2, files will be generated for the document, parts, chapters, sections, and subsections.

You can also remove the table of contents that appears on each page using `--no-display-toc`. If you keep this table of content you can use the option `--toc-depth=N` which specifies the number of levels to include.

If removing the table of contents is not sober enough, you can use the option `--theme=minimal` to get a bare-bone unstyled HTML file, or even `--theme=fragment` to get the content without any header, for later inclusion into a complete HTML document.

They are dozens of other options described briefly when running `plastex --help` and with many more details in the documentation. Typing many command lines options is pretty boring so be sure to read how to put them in a configuration file.

2.2 Rendering both in HTML and pdf

If your document is meant to be rendered only by plasTeX then things are simple, but this is not the most common case. In this section we will assume that you want to use both plasTeX and a pdf-producing compiler such as `pdflatex`, `xelatex` or `lualatex`.

If your document uses LaTeX packages that are not implemented in plasTeX, various things can happen. If you are lucky and your package is simple, plasTeX will manage to load it from its TeX source. If not, then the first thing to consider is whether this complicated package makes sense for the HTML version. Many sophisticated LaTeX packages aim to produce a beautiful pdf file in a way that wouldn't make sense on a web page. In this case, the easiest solution is to have two different preambles. A basic file layout is to have a file `content.tex` containing the actual content of your document and is imported by two small TeX files, `print.tex` for compilation by a pdf producing compiler and `web.tex` for compilation by plasTeX.

```
% Inside file print.tex

\documentclass{article}

\usepackage{fancy_complicated}

\newcommand{\stuff}{horrible hack torturing TeX boxes}

\begin{document}
  \input{content.tex}
\end{document}
```

And then

```
% Inside file web.tex

\documentclass{article}

\newcommand{\stuff}{simple version for the web}

\begin{document}
  \input{content.tex}
\end{document}
```

While this setup is the most versatile, you can try to keep everything in the same file by using the `ifplastex` special conditional. In your TeX file you can include

```
\newif\ifplastex
\plastexfalse
```

This declares a new conditional `\ifplastex`. The second line sets it to false but will be ignored by plasTeX because it recognizes this as a special case. You can then write:

```
\ifplastex
  Some code to be parsed by plasTeX
\else
  Some code for every other TeX compiler
\fi
```

If the above two methods are not enough then you probably want to implement some LaTeX packages as python packages for use by plasTeX. This is beyond the scope of this tutorial but is covered in the documentation.

2.3 Changing the way things are rendered

There are many ways to influence the way plasTeX renders a LaTeX document to a HTML file beyond what the command line options allow to do. The easiest way is to add your own CSS file and tell plasTeX about it using the `extra-css` option, running for instance `plastex --extra-css my_style.css my_document.tex`.

Although CSS tweaks can completely change the appearance of a HTML document, sometimes you also need to change the actual HTML code. The default HTML5 render in plasTeX uses the Jinja templating language. For a very simple example, consider how the default renderer handle a quote environment. The TEX code

```
\begin{quote}
  Cogito ergo sum.
\end{quote}
```

produces the following HTML:

```
<blockquote class="quote">Cogito ergo sum.</blockquote>
```

Let's say you don't like the blockquote tag and want to replace it with a div tag. During parsing by plasTeX, the quote environment will be turned into an instance of the Python `quote` class. Searching for `quote` in the template folder plasTeX/Renderers/HTML5 we find a file `Quotations.jinja2s` containing:

```
name: quote
<blockquote class="quote">{{ obj }}</blockquote>

name: quotation
<blockquote class="quotation">{{ obj }}</blockquote>

name: verse
<blockquote class="verse">{{ obj }}</blockquote>
```

The only Jinja syntax here is the `{{ obj }}` which tells the template engine to render the `obj` variable, here our `quote` instance. Say we would prefer a good old div tag. We can create a modified version of this template file and put it inside some folder, say `my_templates` and then run

```
plastex --extra-templates=my_templates my_doc.tex
```

Note that paths in this option are relative to the current directory.

You can also use this option to create a new theme. For instance you can create a file `my_templates/Themes/my_theme/default-layout.jinja2` taking inspiration from the minimal theme or the default theme depending on whether you prefer starting with a mostly blank slate or tweak the full blown default theme. You can then run

```
plastex --extra-templates=my_templates --theme=my_theme my_doc.tex
```

to use your new layout.