Temporary LaTeX Documents with Orgmode

Rohit Goswami · 2020-06-19 · tools · emacs · workflow · orgmode

> *A post on working with transient TeX templates in `orgmode` without modifying global configurations. This will also serve as a rudimentary introduction to TeX in `orgmode`.*

Background

The sad reality of working in a field dominated by institutional actors which do not care for recognizing software development as a skill is that there are often a lot of ugly LaTeX templates[^fn:1]. In particular, often Universities have arbitrary LaTeX templates from the dark days of 2010 something, which include gratuitous usage of say, `natbib` instead of `biblatex`. In other situations, `.cls` files define separate document classes which are not covered by the `orgmode` defaults and need to be accounted for.

Standard methods

Essentially for the exporter, the document is broken into[^fn:2]:

document_class
: This cannot be changed arbitrarily and has to be a valid element of `org-latex-classes`

preamble
: This section of the document is essentially everything before `\begin{document}` and after `\documentclass{...}`

body
: The rest of the document

We will briefly cover the standard methods of entering TeX in each of these sections, in reverse order since that is the direction in which the intuitive aspect decreases.

In-Body TeX

The method of writing TeX in `orgmode` for the document involves simply writing TeX directly, or wrapping the TeX markup in an export TeX block for font locking[^fn:3]. Essentially, for a document snippet:

```
% #+BEGIN_SRC latex :exports code
\begin{align}
\pi(x) &= \sum_{n=1}^{\infty}\frac{\mu(n)}{n}\Pi(x^{\frac{1}{n}}) \\
       &= \Pi(x) -\frac{1}{2}\Pi(x^{\frac{1}{2}}) - \frac{1}{3}\Pi(x^{\frac{1}{3}}) - \frac{1}
{5}\Pi(x^{\frac{1}{5}}) + \frac{1}{6} \Pi(x^{\frac{1}{6}}) -\cdots,
\end{align}
% #+END_SRC
```

Which will actually be rendered in a real document of course[^fn:4]:

$$\pi(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n}\Pi(x^{\frac{1}{n}}) \qquad = \Pi(x) - \frac{1}{2}\Pi(x^{\frac{1}{2}}) - \frac{1}{3}\Pi(x^{\frac{1}{3}}) - \frac{1}{5}\Pi(x^{\frac{1}{5}}) + \frac{1}{6}\Pi(x^{\frac{1}{6}}) - \cdots,$$

There is also the inline form of writing LaTeX with `@@\sin{x}@@` which is essentially $\sin x$.

Preamble

The main use of the preamble is to either add classes or modify class options for loaded packages like `geometry`. Essentially, for `orgmode`, anything prefixed with `#+LATEX_HEADER:` gets inserted in the preamble.

```
#+LATEX_HEADER: \usepackage{amssymb,amsmath,MnSymbol}
#+LATEX_HEADER: \usepackage{unicode-math}
#+LATEX_HEADER: \usepackage{mathtools}
```

For larger documents, this gets quite annoying for loading packages. We will demonstrate a more aesthetically pleasant form later in this post.

LaTeX classes

Working with document classes is the least intuitive of all TeX manipulations, because for some reason, `#+LATEX_CLASS:` only accepts values defined in `org-latex-classes`.

The standard approach to extending the `orgmode` TeX backend is to add lines like the following in `init.el` or, in my case[^fn:5], `config.org`:

```
(add-to-list 'org-latex-classes
             '("koma-article" "\\documentclass{scrartcl}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
```

This is alright for often used classes like the `koma-*` family of LaTeX document-classes, but it is hardly ideal for one-off TeX templates which are meant for say, grant proposals[^fn:6].

Elisp to the rescue

The core idea is quite simple.

> *Since `orgmode` files are literate documents, and `emacs` is self-documenting and completely programmable, it should be possible to execute code to deterministically set the state of `emacs` before exporting the document to TeX.*

Practically this has a few moving parts. In the following sections, assume that we have a `.cls` file which defines a document-class `foo` with a bunch of packages which conflict with our global configuration.

Adding Document Classes

Instead of adding the code snippet to our global configuration, we will now add it to the document directly with the comments indicating the appropriate environment.

```
;; #+BEGIN_SRC emacs-lisp :exports none  :results none :eval always
(add-to-list 'org-latex-classes
             '("foo" "\\documentclass{foo}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
;; #+END_SRC
```

Where the header arguments simply ensure that the code and result do not show up in the document, and that the chunk is always evaluated by `org-babel`.

Ensuring Purity

Since we want to stick to an external template defined in `foo` **and no other packages** we will need to clear the defaults we lovingly set globally for our convenience.

```
;; #+BEGIN_SRC emacs-lisp :exports none  :results none :eval always
(setq org-latex-packages-alist 'nil)
(setq org-latex-minted-options 'nil) ;; Separately nullify minted
;; #+END_SRC
```

Pretty Packages

For packages we really would like to add, we can now leverage the `elisp` code instead of the ugly `#+LATEX_HEADER:` lines.

```
;; #+BEGIN_SRC emacs-lisp :exports none  :results none :eval always
(setq org-latex-default-packages-alist
  '(("utf8" "inputenc"  t)
    ("normalem" "ulem"  t)
    (""     "mathtools"   t)
    ))
;; #+END_SRC
```

Note that for setting options, we will still need to use the `#+LATEX_HEADER:` syntax.

Automating With Hooks

At this stage, we have a chunk of `elisp` we can manually evaluate with `org-babel` before exporting with `org-latex-export-to-pdf` or `org-latex-export-to-latex`. However, this can get old quickly, so we will instead have a `before-save-hook` to do this for us.

```
# Local Variables:
# before-save-hook: org-babel-execute-buffer
# End:
```

Bonus Hook

In my own configuration, I have a function defined for an `after-save-hook` which generates the TeX file without having me deal with it. For a per-file configuration of this, or globally, the `elisp` is:

```
(defun haozeke/org-save-and-export-latex ()
  (interactive)
  (if (eq major-mode 'org-mode)
    (org-latex-export-to-latex t)))
```

This indirection is required to call the function as a `hook`. The additional `t` allows for asynchronous non blocking exports. Now this can be used as:

```
# Local Variables:
# after-save-hook: haozeke/org-save-and-export-latex
# End:
```

Conclusions

The entire file would look something like this (the `elisp` can be anywhere in the `orgmode` file):

```
;; #+BEGIN_SRC emacs-lisp :exports none  :results none :eval always
(add-to-list 'org-latex-classes
             '("foo" "\\documentclass{foo}"
               ("\\section{%s}" . "\\section*{%s}")
               ("\\subsection{%s}" . "\\subsection*{%s}")
               ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
               ("\\paragraph{%s}" . "\\paragraph*{%s}")
               ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
(setq org-latex-packages-alist 'nil)
(setq org-latex-default-packages-alist
  '(("utf8" "inputenc"  t)
    (""     "minted"   t)
    (""     "rotating"  nil)
    ("normalem" "ulem"  t)
    (""     "mathtools"   t)
    ))
;; #+END_SRC
```

```
#+TITLE: Something
#+AUTHOR: Rohit Goswami
#+OPTIONS: toc:nil \n:nil
#+STARTUP: fninline
#+LATEX_COMPILER: xelatex
#+LATEX_CLASS: foo
#+LATEX_HEADER: \setlength\parindent{0pt}
#+LATEX_HEADER: \addbibresource{./biblio/refs.bib}

Blah blah document $\sin{x}$ stuff

# Local Variables:
# before-save-hook: org-babel-execute-buffer
# after-save-hook: haozeke/org-save-and-export-latex
# End:
```

This method could be extended to essentially resetting all `emacs` variables on a per-file basis (without `file-local-variables` and `dir-local-variables`) or to potentially execute any `elisp` to make `emacs` do things, though I cannot really think of another realistic use-case. The method presented here is really general enough to work with any arbitrary LaTeX `.cls` file or other draconian measures.