

# Org mode $\text{\LaTeX}$ macros (for both HTML and $\text{\LaTeX}$ export)

Brian C. Wells

August 21, 2016

## Contents

This document should be exported to HTML and  $\text{\LaTeX}$  to check that the proper code is generated. A PDF file should also be available, but looks a bit bad because Org mode macros *must* be written on a single line, and some of these macros overfill the line (even in a fairly small font).

This Literate Program is an Org mode setup file that makes it easy to define  $\text{\LaTeX}$  macros that work in *both*  $\text{\LaTeX}$  and HTML export. To "tangle" the source code (`define.org`) for this document to generate the `define.setup` file, type the key sequence `C-c C-v t` in Emacs (with the default keymap). The first line makes sure that the file can be edited in org-mode despite the file being named with an extension of `.setup`.

```
# -*- mode: org -*-
#+MACRO: when-fmt (eval (when (org-export-derived-backend-p org-export-current-backend '$1) "$2"))
#+MACRO: preamble {{{when-fmt(html,\\($1\\))}}}{{{when-fmt(latex,#+LATEX_HEADER: $1)}}}
#+MACRO: define {{{preamble(\\newcommand{$1}$2)}}}
```

## Define the Org mode macros

### when-fmt macro

This is inspired by the `if-latex-else` macro under the ‘Advanced’ heading here: <https://github.com/fniessen/org-macros>. Apparently, Org mode will evaluate Emacs Lisp code in macros, although I have not yet found any documentation that explains *why* this works.

The `when` form is like `if`, except it only returns a string when the condition is true, returning `nil` instead when it is false. We use `when` because we want to perform an action for  $\text{\LaTeX}$  and HTML formats, and we do not

want to assume that the user wants the same behavior for all non-L<sup>A</sup>T<sub>E</sub>X or non-HTML formats.

```
#+MACRO: when-fmt (eval (when (org-export-derived-backend-p org-export-current-backend '$1) "$2"))
```

Since the second parameter \$2 is used inside quotes, it will be necessary to double any backslashes, despite the fact that Org mode macros do not normally require this (except between parameters).

### preamble macro

Using the when-fmt macro, we wrap HTML output in `\(...\)` so that the MathJax library will recognize that it should process them. So long as we only use this to define L<sup>A</sup>T<sub>E</sub>X macros, MathJax will not generate any spurious output. In L<sup>A</sup>T<sub>E</sub>X output, we use the `#+LATEX_HEADER:` Org mode syntax to ensure that it is put in the proper L<sup>A</sup>T<sub>E</sub>X preamble.

```
#+MACRO: preamble {{{when-fmt(html,\\($1\\))}}}{{{when-fmt(latex,#+LATEX_HEADER: $1)}}}
```

### define macro

Using the preamble macro, we specify a macro that uses the L<sup>A</sup>T<sub>E</sub>X `\newcommand` macro to define macros. The first argument is the macro command sequence, and the second argument is whatever L<sup>A</sup>T<sub>E</sub>X code is needed for the definition. Note that we wrap the command sequence in `{...}` automatically, since this is always done. The second parameter is not, however, because it is sometimes necessary to write a number in square brackets [...] when the macro takes parameters.

```
#+MACRO: define {{{preamble(\\newcommand{$1}$2)}}}
```

## Usage Example

Say you want to define a `\mat` command to write the names of matrices, as in `\mat{A}` for **A**.

At the beginning of the file, you should add

```
#+SETUPFILE: define.setup
```

and then you can write

```
{{{define(\\mat,[1]{\\mathbf{#1}})}}}
```

and use it as follows:

`$\mat{A}$`

Again, remember: it is necessary to double the backslashes. And the result looks like this: **A**.