Use of Mako to aid book writing

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Mar 28, 2015

Contents

This document describes the work flow and how we can utilize many nice DocOnce features when writing chapters for a future, potential book project.

1 Use of variables

Mako is the preprocessor that is always run prior to translating DocOnce documents into a specific format. It means that your DocOnce source is actually a computer program where you can use variables and functions.

Writing chapters that can both live their individual lives and be part of a book faces some challenges for which we have some nice solutions in the coming sections.

1.1 How to speak about "this chapter"

In a book you will often need the phrase "this chapter", but his is inappropriate if the chapter is a stand-alone document. Then you would rather say "this document". Similarly, "this book" must read "this document" in a stand-alone chapter. We have resolved this issue by introducing Mako variables `CHAPTER`, `BOOK`, and `APPENDIX` such that you write

```
In this ${BOOK}, the convention is to use boldface for vectors.
```

For this to work, you need to define `CHAPTER`, `BOOK`, and `APPENDIX` as variables on the command line as part of the `doconce format` command:

Terminal

```
Terminal> doconce format pdflatex ch2 --latex_code_style=pyg \
          CHAPTER=document BOOK=document APPENDIX=document
```

When the book is compiled, you do

Terminal

```
Terminal> doconce format pdflatex ch2 --latex_code_style=pyg \
          CHAPTER=chapter BOOK=chapter APPENDIX=chapter
```

The `make.sh` files for found in `doc/src/chapter/make.sh` and `doc/src/book/make.sh` make proper definitions of `CHAPTER`, `BOOK`, and `APPENDIX`.

2 How to make several variants of the text

Sometimes you want to write some text slightly differently if the chapter is a stand-alone document compared to the case when it is part of a book. Mako if tests are ideal for this. Suppose you introduce a Mako variable `ALONE` that is true/defined if the chapter is a stand-alone document and false/undefined if part of a book. Then you can simply write

```
In this
%% if ALONE:
rather small
%% else:
large
%% endif
${BOOK}
```

Running `doconce format` with the option `-DALONE` will turn `ALONE` to true and the output is typically

```
In this rather small document
```

while for a book we skip `-DALONE` as argument to `doconce format`, which makes `ALONE` undefined, and we get the output

```
In this large book
```

Mako variables can be defined/undefined (boolean variables) or be standard strings:

```
%% if SOME_STRING_VARIABLE in ('value1', 'value2'):
some running text
%% endif

...

%% if not SOME_BOOLEAN_VARIABLE:
some other running text
%% else:
yet more different text
%% endif
```

With Mako variables, you can easily comment out large portions of text by testing on some variable you do not intend to define:

```
%% if EXTRA:
This is
text that
will never
appear in the
output.
%% endif
```

Also, it is straightforward to write more than one version of a chapter. For example, you may want to produce a version of a chapter that is tailored to a specific course, while you for general publishing on the Internet want a more general version, and maybe a third version when the chapter is included in a book for the international market. All this is easily done by if tests on appropriately defined Mako variables

```
%% if COURSE == 'IT1713':
# Specific text for a course IT1713
...
%% elif COURSE == 'IT1713b':
# Specific text for a the special IT1713b variant of the course
...
%% elif COURSE == 'general':
# General text when the chapter is a stand-alone document
...
%% elif COURSE == 'book1':
# Text when course is a part of a particular book
...
%% elif COURSE == 'book2':
# Text when course is a part of another book
...
%% endif
```

3 Use of Mako/Python functions

Such if tests are fine to handle larger portions of text. What if you need to have four versions of just one word or very short text? A Mako function, defined as a standard Python function, is then more appropriate.

Here is a definition of a suitable Mako function, defined inside `<%` and `%>` tags:

```
<%
def chversion(text_IT1413, text_IT1713b, text_general,
              text_book1, text_book2):
    if COURSE == 'IT1713':
        return text_IT1413
    elif COURSE == 'IT1713b':
        text_IT1413b
    elif COURSE == 'general':
        return text_general
    elif COURSE == 'book1':
        return text_book1
    elif COURSE == 'book2':
        return text_book2
    else:
        return 'XXX WRONG value of COURSE: %s' % COURSE
%>
```

In the running text you can call `chversion` with five arguments, corresponding to the desired text in the five cases, and when `doconce format` is run, the value of `COURSE` determines which of the five cases that is used. Here is an example on DocOnce text with a function call to `chversion`:

```
It is extremely important to define the term *cure* accurately.
Here we mean ${chversion('handle', 'handle',
'resolve', 'treat', 'resolve')}.
```

You can easily use long multi-line strings as arguments, e.g.,

```
... ${chversion("""
Here comes
a multi-line
string""",
'short string',
'another short string',
"""4th
multi-line
string""",
'5th string')}
...
```

3.1 How to treat two programming languages in the same text

With these ideas, it becomes straightforward to write a book that has its program examples in multiple languages. Introduce `CODE` as the name of the language and use if tests for larger portions of code and text, and a Mako function for shorter inline texts. Copying code from file can also be hidden in a Mako function such that you write `${copyfile('myprog')}` and automatically get it as `src-ch2/python/myprog.py` if `COODE` is `python`, `src-ch2/matlab/myprog.m` if `CODE` is `matlab`, and so forth. The author has successfully co-written such a book[1] [1] for mathematical programming with either Python or Matlab - the version is set when running `doconce format`.

References

[1] S. Linge and H. P. Langtangen. *Programming for Computations*. 2015.

[1] `http://hplgit.github.io/Programming-for-Computations/pub/p4c/index.html`