Doconce Quick Reference

Hans Petter Langtangen[1,2]

[1]Simula Research Laboratory
[2]University of Oslo

Dec 11, 2012

Contents

0.1 Supported Formats 1
0.2 Title, Authors, and Date 2
0.3 Section Types 2
0.4 Inline Formatting 2
0.5 Lists 3
0.6 Comments 4
0.7 Verbatim/Computer Code 5
0.8 LaTeX Mathematics 6
0.9 Figures and Movies 7
0.10 Tables 7
0.11 Labels, References, Citations, and Index 8
0.12 Capabilities of the ”doconce” Program 10
0.13 Exercises 11
0.14 Labels, Index, and Citations 13
0.15 Preprocessing 13
0.16 Resources 14

WARNING: This quick reference is very incomplete!

0.1 Supported Formats

Doconce currently translates files to the following formats:

- LaTeX (format `latex` and `pdflatex`)
- HTML (format `html`)
- reStructuredText (format `rst`)
- plain (untagged) ASCII (format `plain`)

- Sphinx (format `sphinx`)
- (Pandoc extended) Markdown (format `pandoc`)
- Googlecode wiki (format `gwiki`)
- MediaWiki for Wikipedia and Wikibooks (format `mwiki`)
- Creoloe wiki (format `cwiki`)
- Epydoc (format `epydoc`)
- StructuredText (format `st`)

The best supported formats are `latex`, `sphinx`, `html`, and `plain`.

0.2 Title, Authors, and Date

A typical example of giving a title, a set of authors, a date, and an optional table of contents reads

```
TITLE: On an Ultimate Markup Language
AUTHOR: H. P. Langtangen at Center for Biomedical Computing, Simula Research Laboratory
AUTHOR: Kaare Dump Email: dump@cyb.space.com at Segfault, Cyberspace Inc.
AUTHOR: A. Dummy Author
DATE: today
TOC: on
```

The entire title must appear on a single line. The author syntax is

```
name Email: somename@adr.net at institution1 and institution2
```

where the email is optional, the ”at” keyword is required if one or more institutions are to be specified, and the ”and” keyword separates the institutions. Each author specification must appear on a single line. When more than one author belong to the same institution, make sure that the institution is specified in an identical way for each author.

The date can be set as any text different from `today` if not the current date is wanted, e.g., `Feb 22, 2016`.

The table of contents is removed by writing `TOC: off`.

0.3 Section Types

Section type	Syntax
chapter	`========= Heading ========` (9 =)
section	`======= Heading =======` (7 =)
subsection	`===== Heading =====` (5 =)
subsubsection	`=== Heading ===` (3 =)
paragraph	`__Heading.__` (2 _)
abstract	`__Abstract.__` Running text...

Note that abstracts are recognized by starting with `__Abstract.__` at the beginning of a line and ending with three or more = signs of the next heading.

Appendix is supported: just let the heading start with "Appendix: " (this affects only `latex` output, where the appendix formatting is used - all other formats just leave the heading as it is written).

0.4 Inline Formatting

Words surrounded by * are emphasized: `*emphasized words*` becomes *emphasized words*. Similarly, an underscore surrounds words that appear in boldface: `_boldface_` become **boldface**.

0.5 Lists

There are three types of lists: *bullet lists*, where each item starts with *, *enumeration lists*, where each item starts with `o` and gets consqutive numbers, and *description* lists, where each item starts with - followed by a keyword and a colon.

```
Here is a bullet list:

 * item1
 * item2
  * subitem1 of item2
  * subitem2 of item2
 * item3

Note that sublists are consistently indented by one or more blanks..
Here is an enumeration list:

 o item1
 o item2
   may appear on
   multiple lines
  o subitem1 of item2
  o subitem2 of item2
 o item3

And finally a description list:

 - keyword1: followed by
   some text
   over multiple
   lines
 - keyword2:
```

```
   followed by text on the next line
 - keyword3: and its description may fit on one line
```

The code above follows.

Here is a bullet list:

- item1
- item2
  - subitem1 of item2
  - subitem2 of item2
- item3

Note that sublists are indented. Here is an enumeration list:

1. item1
2. item2 may appear on multiple lines
   (a) subitem1 of item2
   (b) subitem2 of item2
3. item3

And finally a description list:

keyword1: followed by some text over multiple lines

keyword2: followed by text on the next line

keyword3: and its description may fit on one line

0.6 Comments

Lines starting with `#` are treated as comments in the document and translated to the proper syntax for comments in the output document. Such comment lines should not appear before LaTeX math blocks, verbatim code blocks, or lists if the formats `rst` and `sphinx` are desired.

When using the Mako preprocessor one can also place comments in the Doconce source file that will be removed by Mako before Doconce starts processing the file. Mako comments are recognized by lines starting with two hashes `##` or by blocks of text inside the comment directives `%<doc>` (beginning) and `<%doc/>` (end).

Inline comments, in the text, that are meant as messages or notes to readers (authors in particular) are often useful and enabled by the syntax

```
[name: running text]
```

where `name` is the name or ID of an author or reader making the comment, and `running text` is the comment. There must be a space after the colon. Running

```
doconce format html mydoc.do.txt --skip_inline_comments
```

removes all such inline comments from the output. This feature makes it easy to turn on and off notes to readers and is frequently used while writing a document.

All inline comments to readers can also be physically removed from the Doconce source if desired:

```
doconce remove_inline_comments mydoc.do.txt
```

This action is appropriate when all issues with such comments are resolved.

0.7 Verbatim/Computer Code

Inline verbatim code is typeset within back-ticks, as in

```
Some sentence with ‘words in verbatim style‘.
```

resulting in Some sentence with `words in verbatim style`.

Multi-line blocks of verbatim text, typically computer code, is typeset in between bc xxx! and ec! directives (which must appear on the beginning of the line). A specification `xxx` indicates what verbatim formatting style that is to be used. Typical values for `xxx` are nothing, `cod` for a code snippet, `pro` for a complete program, `sys` for a terminal session, `dat` for a data file (or output from a program), `Xpro` or `Xcod` for a program or code snipped, respectively, in programming `X`, where `X` may be `py` for Python, `cy` for Cython, `sh` for Bash or other Unix shells, `f` for Fortran, `c` for C, `cpp` for C++, `m` for MATLAB, `pl` for Perl. For output in `latex` one can let `xxx` reflect any defined verbatim environment in the `ptex2tex` configuration file (`.ptex2tex.cfg`). For `sphinx` output one can insert a comment

```
# sphinx code-blocks: pycod=python cod=fortran cppcod=c++ sys=console
```

that maps environments (`xxx`) onto valid language types for Pygments (which is what `sphinx` applies to typeset computer code).

The `xxx` specifier has only effect for `latex` and `sphinx` output. All other formats use a fixed monospace font for all kinds of verbatim output.

Computer code can also be copied from a file:

```
@@@CODE doconce_program.sh
@@@CODE doconce_program.sh  fromto: doconce clean@^doconce split_rst
@@@CODE doconce_program.sh  from-to: doconce clean@^doconce split_rst
```

The `@@@CODE` identifier must appear at the very beginning of the line. The first specification copies the complete file `doconce_program.sh`. The second specification copies from the first line matching the *regular expression* `doconce clean` up to, but not including the line matching the *regular expression* `^doconce split_rst`. The third specification behaves as the second, but the line matching the first regular expression is not copied (aimed at copying text between begin-end comment pair in the file).

The copied line from file are in this example put inside bc shpro! and ec! directives, if a complete file is copied, while the directives become bc shcod! and ec! when a code snippet is copied from file. In general, for a filename extension `.X`, the environment becomes bc Xpro! or bc Xcod! for a complete program or snippet, respectively. The enivorments (`Xcod` and `Xpro`) are only active for `latex` and `sphinx` outout.

Important warnings:

- A code block must come after some plain sentence (at least for successful output in reStructredText), not directly after a section/paragraph heading, table, comment, figure, or movie.
- Verbatim code blocks inside lists can be ugly typeset in some output formats. A more robust approach is to replace the list by paragraphs with headings.

0.8 LaTeX Mathematics

Doconce supports inline mathematics and blocks of mathematics, using standard LaTeX syntax. The output formats `sphinx`, `latex`, and `pdflatex` work with this syntax while all other formats will just display the raw LaTeX code.

Inline expressions are written in the standard LaTeX way with the mathematics surrounded by dollar signs, as in $Ax = b$. To help increase readability in other formats than `sphinx`, `latex`, and `pdflatex`, inline mathematics may have a more human readable companion expression. The syntax is like

```
$\sin(\norm{\bf u})$|$sin(||u||)$
```

That is, the LaTeX expression appears to the left of a vertical bar (pipe symbol) and the more readable expression appears to the right. Both expressions are surrounded by dollar signs. Plain text formats and HTML will applied the expression to the right.

Blocks of LaTeX mathematics are written within bt! and et! ("begin/end TeX") directives. For example,

```
\begin{align*}
\nabla\cdot u &= 0,\\
\nabla\times u & 0.
\end{align*}
```

will appear as

$$\nabla \cdot u = 0,$$
$$\nabla \times u0.$$

One can use `#if FORMAT in ("latex", "pdflatex", "sphinx", "mwiki")` to let the preprocessor choose a block of mathematics in LaTeX format or (`#else`) a modified form more suited for plain text and wiki formats without support for mathematics.

Any LaTeX syntax is accepted, but if output in the `sphinx`, `pandoc`, or `html` formats is important, one must know that these formats does not support many LaTeX constructs. For output both in `latex` and the mentioned formats the following rules are recommended:

- Use only the equation environments `\[`, `\]`, `equation`, `equation*`, `align`, and `align*`.
- Labels in multiple equation environments such as `align` are not (yet) handled by `sphinx` and `pandoc`, so avoid inserting labels and referring to equation labels in `align` environments. Actually, `align*` is the preferred environment for multiple equations.
- LaTeX supports lots of fancy formatting, for example, multiple plots in the same figure, footnotes, margin notes, etc. Allowing other output formats, such as `sphinx`, makes it necessary to only utilze very standard LaTeX and avoid, for instance, more than one plot per figure. However, one can use preprocessor if-tests on the format (typically `# #if FORMAT in ("latex", "pdflatex")`) to include special code for `latex` and `pdflatex` output and more straightforward typesetting for other formats. In this way, one can also allow advanced LaTeX features and fine tuning of resulting PDF document.

LaTeX Newcommands. Text missing...

0.9 Figures and Movies

Figures and movies have almost equal syntax:

```
FIGURE: [relative/path/to/figurefile, width=500] Here goes the caption which must be on a single

MOVIE: [relative/path/to/moviefile, width=500] Here goes the caption which must be on a single li
```

Note the mandatory comma after the figure/movie file.

The figure file can be listed without extension. Doconce will then find the version of the file with the most appropriate extension for the chosen output format. If not suitable version is found, Doconce will convert another format to the desired one.

Movie files can either be a video or a wildcard expression for a series of frames. In the latter case, a simple device in an HTML page will display the individual frame files as a movie.

Combining several image files into one can be done by the `convert` and `montage` programs from the ImageMagick suite:

```
montage file1.png file2.png ... file4.png -geometry +2+2  result.png
montage file1.png file2.png -tile x1 result.png
montage file1.png file2.png -tile 1x result.png

convert -background white file1.png file2.png +append tmp.png
```

Use `+append` for stacking left to right, `-append` for top to bottom. The positioning of the figures can be controlled by `-gravity`.

0.10 Tables

The table in Section 0.3 was written with this syntax:

```
|----------------c--------|------------------c--------------------|
|      Section type       |        Syntax                         |
|----------------l--------|------------------l--------------------|
| chapter                 | ‘========= Heading ========‘ (9 ‘=‘)  |
| section                 | ‘======= Heading =======‘    (7 ‘=‘)  |
| subsection              | ‘===== Heading =====‘        (5 ‘=‘)  |
| subsubsection           | ‘=== Heading ===‘            (3 ‘=‘)  |
| paragraph               | ‘__Heading.__‘               (2 ‘_‘)  |
|-------------------------------------------------------------------|
```

Note that

- Each line begins and ends with a vertical bar (pipe symbol).
- Column data are separated by a vertical bar (pipe symbol).
- There may be horizontal rules, i.e., lines with dashes for indicating the heading and the end of the table, and these may contain characters 'c', 'l', or 'r' for how to align headings or columns. The first horizontal rule may indicate how to align headings (center, left, right), and the horizontal rule after the heading line may indicate how to align the data in the columns (center, left, right).
- If the horizontal rules are without alignment information there should be no vertical bar (pipe symbol) between the columns. Otherwise, such a bar indicates a vertical bar between columns in LaTeX.
- Many output formats are so primitive that heading and column alignment have no effect.

0.11 Labels, References, Citations, and Index

The notion of labels, references, citations, and an index is adopted from LaTeX with a very similar syntax. As in LaTeX, a label can be inserted anywhere, using the syntax

```
label{name}
```

with no backslash preceding the label keyword! It is common practice to choose `name` as some hierarchical name, say `a:b:c`, where `a` and `b` indicate some abbreviations for a section and/or subsection for the topic and `c` is some name for the particular unit that has a label.

A reference to the label `name` is written as

```
ref{name}
```

again with no backslash before `ref`.

Single citations are written as

```
cite{name}
```

where `name` is a logical name of the reference (again, LaTeX writers must not insert a backslash). Bibliography citations often have `name` on the form `Author1_Author2_YYYY`, `Author_YYYY`, or `Author1_etal_YYYY`, where `YYYY` is the year of the publication. Multiple citations at once is possible by separating the logical names by comma:

```
cite{name1,name2,name3}
```

The bibliography is specified by a line `BIBFILE: name_bib.bib, name_bib.rst, name_bib.py`, where `name` is the logical name of the document (the doconce file will then normally have the name `name.do.txt`), and the various files reflect different formattings of the bibliography: '.bib' indicates a BibTeX file, '.rst' a reST-style bibliography, and '.py' a Python list of dictionaries for specifying the entries in the bibliography. The bibliography (as read from file) is inserted where the `BIBFILE` keyword appears.

There is a *generalized referencing* feature in Doconce that allows a reference with `ref` to have one formulation if the label is in the same document and another formulation if the reference is to an item in an external document. The syntax of a generalized reference is

```
ref[internal][cite][external]
```

```
# Example:
As explained in
ref[Section ref{subsec:ex}][in cite{testdoc:12}][a "section":
"testdoc.html#___sec2" in the document
"A Document for Testing Doconce": "testdoc.html" cite{testdoc:12}],
Doconce documents may include movies.
```

The output from a generalized reference is the text `internal` if all `ref{label}` references in `internal` are references to labels in the present document. Otherwise, if cite is non-empty and the format is `latex` or `pdflatex` one assumes that the references in `internal` are to external documents declared by a comment line `# Externaldocuments: testdoc, mydoc` (usually after the title, authors, and date). In this case the output text is `internal cite` and the LaTeX package `xr` is used to handle the labels in the external documents. If none of the two situations above applies, the `external` text will be the output.

Doconce supports creating an index of keywords. A certain keyword is registered for the index by a syntax like (no backslash!)

```
index{name}
```

It is recommended to place any index of this type outside running text, i.e., after (sub)section titles and in the space between paragraphs. Index specifications placed right before paragraphs also gives the doconce source code an indication of the content in the forthcoming text. The index is only produced for the `latex`, `rst`, and `sphinx` formats.

0.12 Capabilities of the "doconce" Program

The `doconce` program can be used for a number of purposes besides transforming a `.do.txt` file to some format. Here is the list of capabilities:

```
Usage: doconce command [optional arguments]
commands: format insertdocstr old2new_format gwiki_figsubst remove_inline_comments lat

doconce format html|latex|pdflatex|rst|sphinx|plain|gwiki|mwiki|cwiki|pandoc|st|epytex

doconce subst [-s -m -x --restore] regex-pattern regex-replacement file1 file2 ...
(-s is the re.DOTALL modifier, -m is the re.MULTILINE modifier,
 -x is the re.VERBOSE modifier, --restore copies backup files back again)

doconce replace from-text to-text file1 file2 ...
(exact text substutition)

doconce replace_from_file file-with-from-to file1 file2 ...
(exact text substitution, but a set of from-to relations)

doconce gwiki_figsubst file.gwiki URL-of-fig-dir

doconce remove_inline_comments file.do.txt

doconce sphinx_dir author='Me and you' title='Quick title' \
    version=0.1 dirname=sphinx-rootdir theme=default \
    file1 file2 file3
(requires sphinx version >= 1.1)
```

```
doconce latin2html file.html

doconce insertdocstr rootdir

doconce clean
(remove all files that the doconce format can regenerate)

doconce latex_header
doconce latex_footer

doconce change_encoding utf-8 latin1 filename
doconce guess_encoding filename

doconce bbl2rst file.bbl
doconce split_rst complete_file.rst
doconce sphinxfix_local_URLs file.rst
doconce split_html complete_file.html
doconce slides_html slide_type complete_file.html
doconce html_colorbullets file1.html file2.html ...

doconce grab   --from[-] from-text [--to[-] to-text] somefile
doconce remove --from[-] from-text [--to[-] to-text] somefile
doconce remove_exercise_answers file_in_some_format
doconce spellcheck [-d .mydict.txt] *.do.txt
doconce ptex2tex mydoc -DMINTED pycod=minted sys=Verbatim \
        dat=\begin{quote}\begin{verbatim};\end{verbatim}\end{quote}

doconce expand_commands file1 file2 ...
doconce combine_images image1 image2 ... output_file
doconce latex_exercise_toc doconcefile.do.txt
doconce list_labels doconcefile.do.txt | latexfile.tex
doconce teamod name
doconce assemble name master.do.txt
```

0.13 Exercises

Doconce supports *Exercise*, *Problem*, and *Project*. These are typeset as ordinary sections and referred to by their section labels. An exercise, problem, or project sections contains certain *elements*:

- a headline at the level of a subsection or subsubsection, containing one of the words ”Exercise:”, ”Problem:”, or ”Project:”, followed by a title (required)
- a label (optional)

- a solution file (optional)
- name of file with a student solution (optional)
- main exercise text (required)
- a short answer (optional)
- a full solution (optional)
- one or more hints (optional)
- one or more subexercises (subproblems, subprojects), which can also contain a text, a short answer, a full solution, name student file to be handed in, and one or more hints (optional)

A typical sketch of a a problem without subexercises goes as follows:

```
===== Problem: Derive the Formula for the Area of an Ellipse =====
label{problem:ellipsearea1}
file=ellipse_area.pdf
solution=ellipse_area1_sol.pdf

Derive an expression for the area of an ellipse by integrating
the area under a curve that defines half of the allipse.
Show each step in the mathematical derivation.

!bhint
Wikipedia has the formula for the curve.
!ehint

!bhint
"Wolframalpha": "http://wolframalpha.com" can perhaps
compute the integral.
!ehint
```

An exercise with subproblems, answers and full solutions has this setup-up:

```
===== Exercise: Determine the Distance to the Moon =====
label{exer:moondist}

Intro to this exercise. Questions are in subexercises below.

!bsubex
Subexercises are numbered a), b), etc.

file=subexer_a.pdf

!bans
```

```
Short answer to subexercise a).
!eans

!bhint
First hint to subexercise a).
!ehint

!bhint
Second hint to subexercise a).
!ehint
!esubex

!bsubex
Here goes the text for subexercise b).

file=subexer_b.pdf

!bhint
A hint for this subexercise.
!ehint

!bsol
Here goes the solution of this subexercise.
!esol
!esubex

!bremarks
At the very end of the exercise it may be appropriate to summarize
and give some perspectives. The text inside the !bremarks-!eremarks
directives is always typeset at the end of the exercise.
!eremarks

!bsol
Here goes a full solution of the whole exercise.
!esol
```

By default, answers, solutions, and hints are typeset as paragraphs.

0.14 Labels, Index, and Citations

0.15 Preprocessing

Doconce documents may utilize a preprocessor, either `preprocess` and/or `mako`. The former is a C-style preprocessor that allows if-tests and including other

files (but not macros with arguments). The `mako` preprocessor is much more advanced - it is actually a full programming language, very similar to Python.

The command `doconce format` first runs `preprocess` and then `mako`. Here is a typical example on utilizing `preprocess` to include another document, "comment out" a large portion of text, and to write format-specific constructions:

```
# #include "myotherdoc.do.txt"

# #if FORMAT in ("latex", "pdflatex")
\begin{table}
\caption{Some words... label{mytab}}
\begin{tabular}{lrr}
\hline\noalign{\smallskip}
\multicolumn{1}{c}{time} & \multicolumn{1}{c}{velocity} & \multicolumn{1}{c}{accelerati
\hline
0.0          & 1.4186       & -5.01        \\
2.0          & 1.376512     & 11.919       \\
4.0          & 1.1E+1       & 14.717624    \\
\hline
\end{tabular}
\end{table}
# #else
  |--------------------------------|
  |time  | velocity | acceleration |
  |--l--------r-----------r--------|
  | 0.0  | 1.4186   | -5.01        |
  | 2.0  | 1.376512 | 11.919       |
  | 4.0  | 1.1E+1   | 14.717624    |
  |--------------------------------|
# #endif

# #ifdef EXTRA_MATERIAL
....large portions of text...
# #endif
```

With the `mako` preprocessor the if-else tests have slightly different syntax. An example document contains some illustrations on how to utilize `mako` (clone the GitHub project and examine the Doconce source and the `doc/src/make.sh` script).

0.16 Resources

- Excellent "Sphinx Tutorial" by C. Reller: "http://people.ee.ethz.ch/ creller/web/tricks/reST.html"