DocOnce Quick Reference

Hans Petter Langtangen[1,2]

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

Oct 6, 2020

Contents

WARNING: This quick reference is very incomplete!

Mission. Enable writing documentation with much mathematics and computer code *once, in one place* and include it in traditional LaTeX books, thesis, and reports, and without extra efforts also make professionally looking web versions with Sphinx or HTML. Other outlets include Google's `blogger.com`, Wikipedia/Wikibooks, IPython notebooks, plus a wide variety of other formats for documents without mathematics and code.

0.1 Supported Formats

DocOnce currently translates files to the following formats:

- LaTeX (format `latex` or `pdflatex`)
- HTML (format `html`)
- Sphinx (format `sphinx`)
- Pandoc-extended or GitHub-flavored Markdown (format `pandoc`)
- IPython notebook (format `ipynb`)
- Matlab notebook (format `matlabnb`)
- MediaWiki (format `mwiki`)
- Googlecode wiki (format `gwiki`)
- Creoloe wiki (format `cwiki`)
- reStructuredText (format `rst`)
- plain (untagged) ASCII (format `plain`)
- Epydoc (format `epydoc`)
- StructuredText (format `st`)

For documents with much code and mathematics, the best (and most supported) formats are `latex`, `pdflatex`, `sphinx`, and `html`; and to a slightly less extent `mwiki` and `pandoc`. The HTML format supports blog posts on Google and Wordpress.

Use a text editor with monospace font!

Some DocOnce constructions are sensitive to whitespace, so you *must* use a text editor with monospace font.

0.2 Emacs syntax support

The file .doconce-mode.el in the DocOnce source distribution gives a "DocOnce Editing Mode" in Emacs. Store the raw version of the file in the home directory and add `(load-file " /.doconce-mode.el")` to the `.emacs` file.

Besides syntax highlighting of DocOnce documents, this Emacs mode provides a lot of shortcuts for setting up many elements in a document:

Emacs key	Action
Ctrl+c f	figure
Ctrl+c v	movie/video
Ctrl+c h1	heading level 1 (section/h1)
Ctrl+c h2	heading level 2 (subsection/h2)
Ctrl+c h3	heading level 2 (subsection/h3)
Ctrl+c hp	heading for paragraph
Ctrl+c me	math environment: `!bt` equation `!et`
Ctrl+c ma	math environment: `!bt` align `!et`
Ctrl+c ce	code environment: `!bc` code `!ec`
Ctrl+c cf	code from file: `@@@CODE`
Ctrl+c table2	table with 2 columns
Ctrl+c table3	table with 3 columns
Ctrl+c table4	table with 4 columns
Ctrl+c exer	exercise outline
Ctrl+c slide	slide outline
Ctrl+c help	print this table

0.3 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 & Dept. of In
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 & institution2
```

where the email is optional, the "at" keyword is required if one or more institutions are to be specified, and the `&` keyword separates the institutions (the keyword `and` works too). 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`.

0.4 Copyright

Copyright for selected authors and/or institutions are easy to insert as part of the `AUTHOR` command. The copyright syntax is

```
{copyright,year1-year2|license}
```

and can be placed after the author or after an institution, e.g.,

```
AUTHOR: name Email: somename@adr.net {copyright,2006-present} at inst1
AUTHOR: name {copyright} at inst1 {copyright}
```

The first line gives `name` a copyright for 2006 up to the present year, while the second line gives copyright to `name` and the institution `inst1` for the present year. The license can be any formulation, but there are some convenient abbreviations for Creative Commons ("public domain") licenses: `CC BY` for Creative Commons Attribution 4.0 license, `CC BY-NC` for Creative Commons Attribution-NonCommercial 4.0 license. For example,

```
AUTHOR: name1 {copyright|CC BY} at institution1
AUTHOR: name2 {copyright|CC BY} at institution2
```

is a very common copyright for the present year with the Attribution license. The copyright must be identical for all authors and institutions.

0.5 Section Types

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

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

The `Exercise:` keyword can be substituted by `Problem:` or `Project:`. A recommended convention is that an exercise is tied to the text, a problem can stand on its own, and a project is a comprehensive problem.

0.6 Inline Formatting

Words surrounded by * are emphasized: `*emphasized words*` becomes *emphasized words*. Similarly, an underscore surrounds words that appear in boldface: `_boldface_` becomes **boldface**. Colored words are also possible: the text

```
‘color{red}{two red words}‘
```

becomes two red words.

Quotations appear inside double backticks and double single quotes:

```
This is a sentence with ‘‘words to be quoted’’.
```

A forced linebreak is specified by `<linebreak>` at the point where the linebreak in the output is wanted.

Footnotes use a label in the text with the footnote text separate, preferably after the paragraph where the footnote appears:

```
Differentiating[^diff2] this equation leads
to a new and much simpler equation.

[^diff2]: More precisely, we apply the divergence
$\nabla\cdot$ on both sides.

Here comes a new paragraph...
```

Non-breaking space is inserted using the tilde character as in LaTeX:

```
This distance corresponds to 7.5~km, which is traveled in $7.5/5$~s.
```

A horizontal rule for separating content vertically, like this:

—

is typeset as four or more hyphens on a single line:

```
---------
```

The `latex`, `pdflatex`, `sphinx`, and `html` formats support em-dash, indicated by three hyphens: `---`. Here is an example:

```
The em-dash is used - without spaces - as alternative to hyphen with
space around in sentences---this way, or in quotes:
*Premature optimization is the root of all evil.*--- Donald Knuth.
```

This text is in the pdflatex rendered as

The em-dash is used - without spaces - as alternative to hyphen with space around in sentences—this way, or in quotes: *Premature optimization is the root of all evil.*— Donald Knuth.

The en-dash consists of two hyphens, either with blanks on both sides – for something in the middle of a sentenc – or in number ranges like 240–249. LaTeX writes are used to and fond of en-dash.

An ampersand, as in Guns & Roses or,Texas A & M, is written as a plain `&` *with space(s) on both sides*. Single upper case letters on each side of `&`, as in

`Texas A {\&} M`, remove the spaces and result in,Texas A & M, while words on both sides of `&`, as in `Guns {\&} Roses`, preserve the spaces: Guns & Roses. Failing to have spaces before and after `&` will result in wrong typesetting of the ampersand in the `html`, `latex`, and `pdflatex` formats.

Emojis, as defined in http://www.emoji-cheat-sheet.com, can be inserted in the text, as (e.g.) `:dizzy_face:` with blank or newline before or after 😵 Only the `pdflatex`, `html`, and `pandoc` output formats translate emoji specifications to images, while all other formats leave the textual specification in the document. The command-line option `--no_emoji` removes all emojis from the output document.

0.7 Lists

There are three types of lists: *bullet lists*, where each item starts with `*`, *enumeration lists*, where each item starts with `o` and gets consecutive 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,
    second line of subitem2
 * item3

Note that sublists are consistently indented by one or more blanks as
shown: bullets must exactly match and continuation lines must start
right below the line above.

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 consistently indented by one or more blanks as shown: bullets must exactly match and continuation lines must start right below the line above.

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

No indentation - except in lists!

DocOnce syntax is sensitive to whitespace. No lines should be indented, only lines belonging to lists. Indented lines may give strange output in some formats.

0.8 Comment lines

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.

Comment lines starting with `##` are not propagated to the output document and can be used for comments that are only of interest in the DocOnce file.

Large portions of text can be left out using Preprocess. Just place `# #ifdef EXTRA` and `# #endif` around the text. The command line option `-DEXTRA` will bring the text alive again.

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.

0.9 Inline comments

Inline comments meant as messages or notes, to authors during development in particular, are 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. The name can contain upper and lower case characters, digits, single quote, + and -, as well as space. Here goes an example.

```
Some running text. [hpl: There must be a space after the colon,
but the running text can occupy multiple lines.]
```

which is rendered as

> Some running text. **hpl 1**: There must be a space after the colon, but the running text can occupy multiple lines.

The inline comments have simple typesetting in most formats, typically boldface name, a comment number, with everything surrounded by parenthesis. However, with LaTeX output and the `--latex_todonotes` option to `doconce format`, colorful margin or inline boxes (using the `todonotes` package) make it very easy to spot the comments.

Running

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

removes all inline comments from the output. This feature makes it easy to turn on and off notes to authors during the development of the document.

All inline comments to readers can also be physically removed from the DocOnce source by

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

Inline comments can also be used to markup edits. There are add, delete, and replacement comments for editing:

```
[add: ,]
[add: .]
[add: ;]
[del: ,]
[del: ,]
[del: .]
[del: ;]
[add: some text]
[del: some text]
[edit: some text -> some replacement for text]
[name: some text -> some replacement for text]
```

For example, the text

```
First consider a quantity $Q$. Without loss of generality, we assume
$Q>0$. There are three, fundamental, basic property of $Q$.
```

can be edited as

```
First[add: ,] consider [edit: a quantity -> the flux]
[del: $Q$. Without loss of generality,
we assume] $Q>0$. There are three[del: ,] fundamental[del: , basic]
[edit: property -> properties] of $Q$. [add: These are not
important for the following discussion.]
```

which in the pdflatex output format results in

> First, (**edit 2**: add comma) consider (edit 3:) ~~a quantity~~ the flux (**edit 4**:) ~~Q. Without loss of generality, we assume~~ $Q > 0$. There are three (**edit 5**: delete comma) fundamental(**edit 6**:) ~~, basic~~ (edit 7:) ~~property~~ properties of Q. (**edit 8**:) These are not important for the following discussion.

To implement these edits, run

```
Terminal> doconce apply_edit_comments mydoc.do.txt
```

0.10 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.

Here is an example of computer code (see the source of this document for exact syntax):

```
from numpy import sin, cos, exp, pi

def f(x, y, z, t):
    return exp(-t)*sin(pi*x)*sin(pi*y)*cos(2*pi*z)
```

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
@@@CODE doconce_program.sh  envir=shpro fromto: name=@
```

The `@@@CODE` identifier must appear at the very beginning of the line. The first line copies the complete file `doconce_program.sh`. The second line 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 line behaves as the second, but the line matching the first regular expression is not copied (this construction often used for copying text between begin-end comment pair in the file).

The copied lines 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`, `pdflatex`, html‘, and `sphinx` outout. The fourth line above specifies the code environment explicitly (`envir=shpro`) such that it indicates a complete shell program (`shpro`) even if we copy a part of the file (here from `name=` until the end of the file). Copying a part will by default lead to `!bc shcod`, which indicates a code snippet that normally needs more code to run properly.

The `--code_prefix=text` option adds a path `text` to the filename specified in the `@@@CODE` command (URLs work). For example

```
@@@CODE src/myfile.py
```

and `--code_prefix=http://some.place.net`, the file

```
http://some.place.net/src/myfile.py
```

will be included. If source files have a header with author, email, etc., one can remove this header by the option `'--code_skip_until=# ---`. The lines up to and including (the first) `# ---` will then be excluded.

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.

0.11 LaTeX Mathematics

DocOnce supports inline mathematics and blocks of mathematics, using standard LaTeX syntax. The output formats `html`, `sphinx`, `latex`, pdflatex‘, `pandoc`, and `mwiki` 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.

Blocks of LaTeX mathematics are written within `!bt` and `!et` (begin/end TeX) directives starting on the beginning of a line:

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

This LaTeX code gets rendered as

$$\nabla \cdot \boldsymbol{u} = 0,$$
$$\nabla \times \boldsymbol{u} = 0.$$

Here is a single equation:

```
!bt
\[ \frac{\partial\pmb{u}}{\partial t} + \pmb{u}\cdot\nabla\pmb{u} = 0.\]
!et
```

which results in

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = 0.$$

LaTeX Newcommands. The author can define `newcommand` statements in files with names `newcommands*.tex`. Such commands should only be used for mathematics (other LaTeX constructions are only understood by LaTeX itself). The convention is that `newcommands_keep.tex` contains the newcommands that are kept in the document, while those in `newcommands_replace.tex` will be replaced by their full LaTeX code. This conventions helps make readable documents in formats without LaTeX support. For `html`, `sphinx`, `latex`, `pdflatex`, `mwiki`, `ipynb`, and `pandoc`, the mathematics in newcommands is rendered

nicely anyway. If you desire `newcommand` outside LaTeX mathematics, simply use a Mako variable or a Mako function (which will be much more flexible and powerful).

0.12 Writing Guidelines (Especially for LaTeX Users!)

LaTeX writers often have their own writing habits with use of their own favorite LaTeX packages. DocOnce is a much simpler format and corresponds to writing in quite plain LaTeX and making the ascii text look nice (be careful with the use of white space!). This means that although DocOnce has borrowed a lot from LaTeX, there are a few points LaTeX writers should pay attention to. Experience shows that these points are so important that we list them *before* we list typical DocOnce syntax!

Any LaTeX syntax in mathematical formulas is accepted when DocOnce translates the text to LaTeX, but if output in the `sphinx`, `pandoc`, `mwiki`, `html`, or `ipynb` formats is also important, one should follow the rules below.

- AMS LaTeX mathematics is supported, also for the `html`, `sphinx`, and `ipynb` formats.
- If you want LaTeX math blocks to work with `latex`, `html`, `sphinx`, `markdown`, and `ipynb`, it is recommended to use only the following equation environments: `\[ ... \]`, `equation*`, `equation`, `align*`, `align`. `alignat*`, `alignat`. Other environments, such as `split`, `multiline`, `gather` are supported in modern MathJax in HTML and Sphinx, but may face rendering problems (to a larger extent than `equation` and `align`). DocOnce performs extensions to `sphinx`, `ipynb`, and other formats such that labels in `align` and `alignat` environments work well. If you face problems with fancy LaTeX equation environments in web formats, try rewriting with plain `align`, `nonumber`, etc.
- Do not use comments inside equations.
- Newcommands in mathematical formulas are allowed, but not in the running text. Newcommands must be defined in files with names `newcommands*.tex`. Use `\newcommands` and not `\def`. Each newcommand must be defined on a single line. Use Mako functions if you need macros in the running text.
- Use labels and refer to them for sections, figures, movies, and equations only. MediaWiki (`mwiki`) does not support references to equations.
- Spaces are not allowed in labels.
- There is just one `ref` command (no `\eqref` for equations) and references to equations must use parentheses. Never use the tilde ~ (non-breaking space) character before references to figures, sections, etc., but tilde is allowed for references to equations.

- Never use `\pageref` as pages are not a concept in web documents (there is only a `ref` command in DocOnce and it refers to labels).
- Only figures and movies are floating elements in DocOnce, all other elements (code, tables, algorithms) must appear *inline* without numbers or labels for reference[1] (refer to inline elements by a section label). The reason is that floating elements are in general not used in web documents, but we made an exception with figures and movies.
- Keep figure captions shorts as they are used as references in the Sphinx format. Avoid inline mathematics since Sphinx will strip it away in the figure reference. (Many writing styles encourage rich captions that explains everything about the figure, but this will work well only in the HTML and LaTeX formats.)
- You cannot use `subfigure` to combine several image files in one figure, but you can combine the files to one file using the `doconce combine_images` tool. Refer to individual image files in the caption or text by (e.g.) "left" and "right", or "upper left", "lower right", etc.
- Footnotes can be used as usual in LaTeX, but some HTML formats are not able to display mathematics or inline verbatim or other formatted code (emphasis, boldface, color) in footnotes - have that in mind.
- Use plain `cite` for references (e.g., `\citeauthor` has no counterpart in DocOnce). The bibliography must be prepared in the Publish format, but import from (clean) BibTeX is possible.
- Use `idx` for index entries, but put the definitions between paragraphs, not inside them (required by Sphinx).
- Use the `\bm` command (from the `bm` package, always included by DocOnce) for boldface in mathematics.
- Make sure all ordinary text starts in column 1 on each line. Equations can be indented. The `\begin{}` and `\end{}` directives should start in column 1.
- If you depend on various LaTeX environments for your writings, you have to give up these, or implement *user-defined environments* in DocOnce. For instance, examples are normally typeset as subsections in DocOnce, but can also utilize a user-defined example environment. Learn about the exercise support in DocOnce for typesetting exercises, problems, and projects.

[1] There is an exception: by using *user-defined environments* within `!bu-name` and `!eu-name` directives, it is possible to label any type of text and refer to it. For example, one can have environments for examples, tables, code snippets, theorems, lemmas, etc. One can also use Mako functions to implement enviroments.

- Learn about the preprocessors Preprocess and Mako - these are smart tools for, e.g., commenting out/in large portions of text and creating macros.
- Use *generalized references* when referring to companion documents that may later become part of this document (or migrated out of this document).
- Follow recommendations for DocOnce books if you plan to write a book.

Use the preprocessor to tailor output.

If you really need special LaTeX constructs in the LaTeX output from DocOnce, you may use use preprocessor if-tests on the format (typically `#if FORMAT in ("latex", "pdflatex")`) to include such special LaTeX code. With an else clause you can easily create corresponding constructions for other formats. This way of using Preprocess or Mako allows advanced LaTeX features, or HTML features for the HTML formats, and thereby fine tuning of the resulting document. More tuning can be done by automatic editing of the output file (e.g., `.tex` or `.html`) produced by DocOnce using your own scripts or the `doconce replace` and `doconce subst` commands.

Autotranslation of LaTeX to DocOnce?

The tool `doconce latex2doconce` may help you translating LaTeX files to DocOnce syntax. However, if you use computer code in floating list environments, special packages for typesetting algorithms, example environments, `subfigure` in figures, or a lot of newcommands in the running text, there will be need for a lot of manual edits and adjustments.

For examples, figure environments can be translated by the program `doconce latex2doconce` only if the label is inside the caption and the figure is typeset like

```
\begin{figure}
  \centering
  \includegraphics[width=0.55\linewidth]{figs/myfig.pdf}
  \caption{This is a figure. \labe{myfig}}
\end{figure}
```

If the LaTeX is consistent with respect to placement of the label, a simple script can autoedit the label inside the caption, but many LaTeX writers put the label at different places in different figures, and then it becomes more difficult to autoedit figures and translate them to the DocOnce `FIGURE:` syntax.

Tables are hard to interpret and translate, because the headings and caption can be typeset in many different ways. The type of table that is recognized looks like

```
\begin{table}
\caption{Here goes the caption.}
\begin{tabular}{lr}
\hline
\multicolumn{1}{c}{$v_0$} & \multicolumn{1}{c}{$f_R(v_0)$}\\\hline
1.2 & 4.2\1.1 & 4.0\0.9 & 3.7
\hline
\end{tabular}
\end{table}
```

Recall that table captions do not make sense in DocOnce since tables must be inlined and explained in the surrounding text.

Footnotes are also problematic for `doconce latex2doconce` since DocOnce footnotes must have the explanation outside the paragraph where the footnote is used. This calls for manual work. The translator from LaTeX to DocOnce will insert `_PROBLEM_` and mark footnotes. One solution is to avoid footnotes in the LaTeX document if fully automatic translation is desired.

0.13 Hyperlinks

Links use either a link text or the raw URL:

```
Here is some "some link text": "http://some.net/address"
(as in "search google": "http://google.com")
or just the raw address: URL: "http://google.com".

Links to files typeset in verbatim mode applies backtics:
"‘myfile.py‘": "http://some.net/some/place/myfile.py".

Mail addresses works too: send problems to
"‘hpl@simula.no‘": "mailto:hpl@simula.no"
or just "send mail": "mailto:hpl@simula.no".
```

0.14 Figures and Movies

Figures and movies have almost equal syntax:

```
FIGURE: [relative/path/to/figurefile, width=500 frac=0.8] Here goes the caption which must be on a s

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

Note three important syntax details:

1. A mandatory comma after the figure/movie filename,
2. no comments between `width`, `height`, and `frac` and no spaces around the = characters,

3. all of the command must appear on a single line,

4. there must be a blank line after the command.

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 needed one.

The caption is optional. If omitted, the figure will be inlined (meaning no use of any figure environment in HTML or LaTeX formats). The `width` and `height` parameters affect HTML formats (`html`, `rst`, `sphinx`), while `frac` is the width of the image as a fraction of the total text width in the `latex` and `pdflatex` formats.

The command-line options `--fig_prefix=...` and `--mov_prefix=...` can be used to add a path (can be a URL) to all figure and movie files, respectively. This is useful when including DocOnce documents in other DocOnce documents such that the text is compiled in different directories (with different paths to the figure directory).

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

```
doconce combine_images image1 image2 ... output_image
```

This command applies `montage` or PDF-based tools to combine the images to get the highest quality.

YouTube and Vimeo movies will be embedded in `html` and `sphinx` documents and otherwise be represented by a link. The syntax is

```
MOVIE: [http://www.youtube.com/watch?v=_O7iUiftbKU, width=420 height=315] YouTube movie.

MOVIE: [http://vimeo.com/55562330, width=500 height=278] Vimeo movie.
```

The latter results in

http://vimeo.com/55562330

Movie 1: Vimeo movie.

0.15 Tables

The table in Section 0.5 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 must be a blank line before and after the table.
- 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). One can also use `X` for potentially very wide text that must be wrapped and left-adjusted (will only affect `latex` and `pdflatex` where the `tabularx` package is then used; `X` means `l` in all other formats).
- 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.

A quick way of generating tables is to place all the entries in a file with comma as separator (a CSV file) and then run the utility `doconce csv2table` to create a table in the DocOnce format.

The command-line option `-tables2csv` (to `doconce format`) makes DocOnce dump each table to CSV format in a file `table_X.csv`, where `X` is the table number. This feature makes it easy to load tables into spreadsheet programs for further analysis.

DocOnce tables can be efficiently made directly from data in CSV files.

```
Terminal> doconce csv2table mydata.csv > mydata_table.do.txt
```

Now we can do `# #include "mydata_table.do.txt"` in the DocOnce source file or simply copy the table in `mydata_table.do.txt` into the DocOnce file.

0.16 Labels and References

The notion of labels and references (as well as bibliography and 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, using a delimiter like `:` or `_` between (e.g.) section, subsection, and topic.

A reference to the label `name` is written as

```
ref{name}
```

again with no backslash before `ref`.

Use labels for sections and equations only, and preceed the reference by "Section" or "Chapter", or in case of an equation, surround the reference by parenthesis.

0.17 Citations and Bibliography

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: papers.pub`, where `papers.pub` is a publication database in the Publish format. BibTeX `.bib` files can easily be combined to a Publish database (which DocOnce needs to create bibliographies in other formats than LaTeX).

0.18 Generalized References

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. This construction makes it easy to work with many small, independent documents in parallel with a book assembly of some of the small elements. The syntax of a generalized reference is

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

with a specific example being

```
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 references with `ref` in the text `internal` are references to labels defined 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` followed by `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.

0.19 Index of Keywords

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`, `pdflatex`, `rst`, and `sphinx` formats.

0.20 Capabilities of The Program `doconce`

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:

```
DocOnce version 1.5.2 (from /home/amarin/doconce/venv/lib/python3.6/site-packages/DocOnce-1.5.2-py3.6
Usage: doconce command [optional arguments]
commands: help format find subst replace remove spellcheck apply_inline_edits capitalize change_encoc

doconce format html|latex|pdflatex|rst|sphinx|plain|gwiki|mwiki|
             cwiki|pandoc|st|epytext dofile
# transform doconce file to another format

doconce subst [-s -m -x --restore] regex-pattern \
        regex-replacement file1 file2 ...
# substitute a phrase by another using regular expressions (in this example -s is the re.DOTALL modi

doconce replace from-text to-text file1 file2 ...
# replace a phrase by another literally (exact text substitution)

doconce replace_from_file file-with-from-to-replacements file1 file2 ...
# replace using from and to phrases from file

doconce find expression
# search for a (regular) expression in all .do.txt files in the current directory tree (useful when r

doconce include_map mydoc.do.txt
# print an overview of how various files are included in the root doc

doconce expand_mako mako_code_file funcname file1 file2 ...
# replace all mako function calls by the ‘results of the calls
```

```
doconce remove_inline_comments dofile
# remove all inline comments in a doconce file

doconce apply_inline_edits
# apply all edits specified through inline comments

doconce sphinx_dir copyright='John Doe' title='Long title' \
        short_title="Short title" version=0.1 intersphinx \
        /path/to/mylogo.png dofile
# create a directory for the sphinx format (requires sphinx version >= 1.1)

doconce format sphinx complete_file
doconce split_rst complete_file
doconce sphinx_dir complete_file
python automake_sphinx.py
# split a sphinx/rst file into parts according to !split commands

doconce insertdocstr rootdir
# walk through a directory tree and insert doconce files as docstrings in *.p.py files

doconce lightclean
# remove all redundant files (keep source .do.txt and results: .pdf, .html, sphinx- dirs, .mwiki, .ip

doconce clean
# remove all files that the doconce can regenerate

doconce change_encoding utf-8 latin1 dofile
# change encoding

doconce guess_encoding filename
# guess the encoding in a text

doconce find_nonascii_chars file1 file2 ...
# find non-ascii characters in a file

doconce split_html complete_file.html
# split an html file into parts according to !split commands

doconce slides_html slide_type complete_file.html
# create HTML slides from a (doconce) html file

doconce slides_beamer complete_file.tex
# create LaTeX Beamer slides from a (doconce) latex/pdflatex file

doconce slides_markdown complete_file.md remark --slide_style=light
# create Remark slides from Markdown

doconce html_colorbullets file1.html file2.html ...
# replace bullets in lists by colored bullets

doconce extract_exercises tmp_mako__mydoc
# extract all exercises (projects and problems too)

doconce grab --from[-] from-text [--to[-] to-text] file > result
# grab selected text from a file

doconce remove --from[-] from-text [--to[-] to-text] file > result
# remove selected text from a file

doconce grep FIGURE|MOVIE|CODE dofile
# list all figure, movie or included code files

doconce spellcheck [-d .mydict.txt] *.do.txt
```

```
# run spellcheck on a set of files

doconce ptex2tex mydoc -DMINTED pycod=minted sys=Verbatim \
        dat=\begin{quote}\begin{verbatim};\end{verbatim}\end{quote}
# transform ptex2tex files (.p.tex) to ordinary latex file and manage the code environments

doconce md2html file.md
# make HTML file via pandoc from Markdown (.md) file

doconce md2latex file.md
# make LaTeX file via pandoc from Markdown (.md) file

doconce combine_images image1 image2 ... output_file
# combine several images into one

doconce latex_problems mydoc.log [overfull-hbox-limit]
# report problems from a LaTeX .log file

doconce list_fig_src_files *.do.txt
# list all figure files, movie files, and source code files needed

doconce list_labels myfile
# list all labels in a document (for purposes of cleaning them up)

doconce ref_external mydoc [pubfile]
# generate script for substituting generalized references

doconce linkchecker *.html
# check all links in HTML files

doconce capitalize [-d .mydict.txt] *.do.txt
# change headings from "This is a Heading" to "This is a heading"

doconce latex2doconce latexfile
# translate a latex document to doconce (requires usually manual fixing)

doconce latex_dislikes latexfile
# check if there are problems with translating latex to doconce

doconce ipynb2doconce notebookfile
# translate an IPython/Jupyter notebook to doconce

doconce pygmentize myfile [pygments-style]
# typeset a doconce document with pygments (for pretty print of doconce itself)

doconce makefile docname doconcefile [html sphinx pdflatex ...]
# generate a make.py script for translating a doconce file to various formats

doconce diff file1.do.txt file2.do.txt [diffprog]
# find differences between two files (diffprog can be difflib, diff, pdiff, latexdiff, kdiff3, diffus

doconce gitdiff file1 file2 file3 ...
# find differences between the last two Git versions of several files

doconce csv2table somefile.csv
# convert csv file to doconce table format

doconce sphinxfix_local_URLs file.rst
# edit URLs to local files and place them in _static

doconce latin2html file.html
# replace latex-1 (non-ascii) characters by html codes

doconce fix_bibtex4publish file1.bib file2.bib ...
```

```
# fix common problems in bibtex files for publish import

doconce latex_header
# print the header (preamble) for latex file

doconce latex_footer
# print the footer for latex files

doconce expand_commands file1 file2 ...
# expand short cut commands to full form in files

doconce latex_exercise_toc myfile
# insert a table of exercises in a latex file myfile.p.tex
```

0.21 Exercises

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

- a headline at the level of a subsection containing one of the words "Exercise:", "Problem:", "Project:", or "Example:", 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
```

If the exercise type (Exercise, Problem, Project, or Example) is enclosed in braces, the type is left out of the title in the output. For example, the if the title line above reads

```
===== {Problem}: Derive the Formula for the Area of an Ellipse =====
```

the title becomes just "Derive the ...".

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‘ and ‘!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. The command-line arguments `--without_answers` and `--without_solutions` turn off output of answers and solutions, respectively, except for examples. The command line options `--answers_at_end` and `--solutions_at_end` write all answers and solutions to exercises to a separate section in the end of the document, respectively. Combine with `--without_answers` and `--without_solutions` to remove answers and solutions from the main text.

The commands `!anshide` and `!solhide` can be used to hide from the main text answers and solutions, respectively, until the `!ansoff` and `!soloff` commands are encountered. Similarly, the `!ansdocend` and `!soldocend` commands move answers and solutions to the end of the book.

0.22 Environments

DocOnce environments start with `!benvirname` and end with `!eenvirname`, where `envirname` is the name of the environment. Here is a listing of the environments:

- `c`: computer code (or verbatim text)
- `t`: math blocks with LaTeX syntax
- `subex`: sub-exercise
- `ans`: short answer to exercise or sub-exercise
- `sol`: full solution to exercise or sub-exercise
- `hint`: multiple help items in an exercise or sub-exercise
- `quote`: indented text
- `notice`, `summary`, `warning`, `question`: admonition boxes with custom title, special icon, and (sometimes) background color
- `block`, `box`: simpler boxes (`block` may have title but never any icon)
- `pop`: text to gradually pop up in slide presentations
- `slidecell`: indication of cells in a grid layout for elements on a slide

In addition, the user can define new environments `!bc-name` as explained in the manual.

0.23 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}{acceleration} \\
\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).

0.24 Resources

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