

Doconce: Document Once, Include Anywhere

Author: Hans Petter Langtangen

Date: Mar 5, 2013

- When writing a note, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, wiki, Word/OpenOffice, LaTeX, HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?
- Do you need to write documents in varying formats but find it difficult to remember all the typesetting details of various formats like [LaTeX](#), [HTML](#), [reStructuredText](#), [Sphinx](#), and [wiki](#)? Would it be convenient to generate the typesetting details of a particular format from a very simple text-like format with minimal tagging?
- Do you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one format, stored in one place, and include it anywhere?

If any of these questions are of interest, you should keep on reading.

What Does Doconce Look Like?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the formatting. Here are some examples.

- Bullet lists arise from lines starting with `*`.
- *Emphasized words* are surrounded by `*`.
- **Words in boldface** are surrounded by underscores.
- Words from computer code are enclosed in back quotes and then typeset `verbatim` (in a monospace font).
- Section headings are recognized by equality (=) signs before and after the title, and the number of = signs indicates the level of the section: 7 for main section, 5 for subsection, and 3 for subsubsection.
- Paragraph headings are recognized by a double underscore before and after the heading.
- The abstract of a document starts with *Abstract* as paragraph heading, and all text up to the next heading makes up the abstract,
- Blocks of computer code can easily be included by placing `!bc` (begin code) and `!ec` (end code) commands at separate lines before and after the code block.
- Blocks of computer code can also be imported from source files.
- Blocks of LaTeX mathematics can easily be included by placing `!bt` (begin TeX) and `!et` (end TeX) commands at separate lines before and after the math block.

- There is support for both LaTeX and text-like inline mathematics.
- Figures and movies with captions, simple tables, URLs with links, index list, labels and references are supported.
- Invisible comments in the output format can be inserted throughout the text.
- Visible comments can be inserted so that authors and readers can comment upon the text (and at any time turn on/off output of such comments).
- There is an exercise environment with many advanced features.
- With a preprocessor, Preprocess or Mako, one can include other documents (files) and large portions of text can be defined in or out of the text.
- With Mako one can also have Python code embedded in the Doconce document and thereby parameterize the text (e.g., one text can describe programming in two languages).

Here is an example of some simple text written in the Doconce format:

```
===== A Subsection with Sample Text =====
label{my:first:sec}
```

Ordinary text looks like ordinary text, and the tags used for `_boldface_` words, `*emphasized*` words, and `'computer'` words look natural in plain text. Lists are typeset as you would do in email,

```
* item 1
* item 2
* item 3
```

Lists can also have automatically numbered items instead of bullets,

```
o item 1
o item 2
o item 3
```

URLs with a link word are possible, as in `"hpl": "http://folk.uio.no/hpl"`. If the word is URL, the URL itself becomes the link name, as in `"URL": "tutorial.do.txt"`.

References to sections may use logical names as labels (e.g., a `"label"` command right after the section title), as in the reference to Section `ref{my:first:sec}`.

Doconce also allows inline comments of the form `[name: comment]` (with a space after `'name:'`), e.g., such as `[hpl: here I will make some remarks to the text]`. Inline comments can be removed from the output by a command-line argument (see Section `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

time	velocity	acceleration
0.0	1.4186	-5.01
2.0	1.376512	11.919
4.0	1.1E+1	14.717624

lines beginning with # are comment lines

The Doconce text above results in the following little document:

A Subsection with Sample Text

Ordinary text looks like ordinary text, and the tags used for **boldface** words, *emphasized* words, and computer words look natural in plain text. Lists are typeset as you would do in an email,

- item 1
- item 2
- item 3

Lists can also have numbered items instead of bullets, just use an o (for ordered) instead of the asterisk:

1. item 1
2. item 2
3. item 3

URLs with a link word are possible, as in [hpl](#). If the word is URL, the URL itself becomes the link name, as in [tutorial.do.txt](#).

References to sections may use logical names as labels (e.g., a “label” command right after the section title), as in the reference to the section [A Subsection with Sample Text](#).

Doconce also allows inline comments such as (**hpl**: here I will make some remarks to the text) for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see the section [From Doconce to Other Formats](#) for an example).

Tables are also supported, e.g.,

time	velocity	acceleration
0.0	1.4186	-5.01
2.0	1.376512	11.919
4.0	1.1E+1	14.717624

Mathematics and Computer Code

Inline mathematics, such as $v = \sin(x)$, allows the formula to be specified both as LaTeX and as plain text. This results in a professional LaTeX typesetting, but in other formats the text version normally looks better than raw LaTeX mathematics with backslashes. An inline formula like $v = \sin(x)$ is typeset as:

```
$\nu = \sin(x)$| $v = \sin(x)$ $
```

The pipe symbol acts as a delimiter between LaTeX code and the plain text version of the formula. If you write a lot of mathematics, only the output formats `latex`, `pdflatex`, `html`, `sphinx`, and `pandoc` are of interest and all these support inline LaTeX mathematics so then you will naturally drop the pipe symbol and write just:

```
$\nu = \sin(x)$
```

However, if you want more textual formats, like plain text or reStructuredText, the text after the pipe symbol may help to make the math formula more readable if there are backslashes or other special LaTeX symbols in the LaTeX code.

Blocks of mathematics are typeset with raw LaTeX, inside `!bt` and `!et` (begin TeX, end TeX) instructions:

```
!bt
\begin{align}
\{\partial u \over \partial t\} &= \nabla^2 u + f, \text{label{myeq1}} \\
\{\partial v \over \partial t\} &= \nabla \cdot (q(u) \nabla v) + g \\
\end{align}
!et
```

The result looks like this:

```
\begin{align}
\{\partial u \over \partial t\} &= \nabla^2 u + f, \text{label{myeq1}} \\
\{\partial v \over \partial t\} &= \nabla \cdot (q(u) \nabla v) + g \\
\end{align}
```

Of course, such blocks only looks nice in formats with support for LaTeX mathematics, and here the `align` environment in particular (this includes `latex`, `pdflatex`, `html`, and `sphinx`). The raw LaTeX syntax appears in simpler formats, but can still be useful for those who can read LaTeX syntax.

You can have blocks of computer code, starting and ending with `!bc` and `!ec` instructions, respectively:

```
!bc pycod
from math import sin, pi
def myfunc(x):
    return sin(pi*x)

import integrate
I = integrate.trapezoidal(myfunc, 0, pi, 100)
!ec
```

Such blocks are formatted as:

```

from math import sin, pi
def myfunc(x):
    return sin(pi*x)

import integrate
I = integrate.trapezoidal(myfunc, 0, pi, 100)

```

A code block must come after some plain sentence (at least for successful output to sphinx, rst, and ASCII-close formats), not directly after a section/paragraph heading or a table.

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!).

Another document can be included by writing `# #include "mynote.do.txt"` at the beginning of a line. Doconce documents have extension `do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you plain text editing capabilities.

Macros (Newcommands), Cross-References, Index, and Bibliography

Doconce supports a type of macros via a LaTeX-style *newcommand* construction. The newcommands defined in a file with name `newcommand_replace.tex` are expanded when Doconce is filtered to other formats, except for LaTeX (since LaTeX performs the expansion itself). Newcommands in files with names `newcommands.tex` and `newcommands_keep.tex` are kept unaltered when Doconce text is filtered to other formats, except for the Sphinx format. Since Sphinx understands LaTeX math, but not newcommands if the Sphinx output is HTML, it makes most sense to expand all newcommands. Normally, a user will put all newcommands that appear in math blocks surrounded by `!bt` and `!et` in `newcommands_keep.tex` to keep them unchanged, at least if they contribute to make the raw LaTeX math text easier to read in the formats that cannot render LaTeX. Newcommands used elsewhere throughout the text will usually be placed in `newcommands_replace.tex` and expanded by Doconce. The definitions of newcommands in the `newcommands*.tex` files *must* appear on a single line (multi-line newcommands are too hard to parse with regular expressions).

Recent versions of Doconce also offer cross referencing, typically one can define labels below (sub)sections, in figure captions, or in equations, and then refer to these later. Entries in an index can be defined and result in an index at the end for the LaTeX and Sphinx formats. Citations to literature, with an accompanying bibliography in a file, are also supported. The syntax of labels, references, citations, and the bibliography closely resembles that of LaTeX, making it easy for Doconce documents to be integrated in LaTeX projects (manuals, books). For further details on functionality and syntax we refer to the `doc/manual/manual.do.txt` file (see the [demo page](#) for various formats of this document).

From Doconce to Other Formats

Transformation of a Doconce document `mydoc.do.txt` to various other formats applies the script `doconce format`:

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

or just:

```
Terminal> doconce format format mydoc
```

Generating a makefile

Producing HTML, Sphinx, and in particular LaTeX documents from Doconce sources requires a few commands. Often you want to produce several different formats. The relevant commands should then be placed in a script that acts as a “makefile”.

The `doconce makefile` can be used to automatically generate such a makefile, more precisely a Bash script `make.sh`, which carries out the commands explained below. If our Doconce source is in `main_myproj.do.txt`, we run:

```
doconce makefile main_myproj html pdflatex sphinx
```

to produce the necessary output for generating HTML, pdfLaTeX, and Sphinx. Usually, you need to edit `make.sh` to really fit your needs. Some examples lines are inserted as comments to show various options that can be added to the basic commands. A handy feature of the generated `make.sh` script is that it inserts checks for successful runs of the `doconce format` commands, and if something goes wrong, the `make.sh` exits.

Preprocessing

The `preprocess` and `mako` programs are used to preprocess the file, and options to `preprocess` and/or `mako` can be added after the filename. For example:

```
Terminal> doconce format latex mydoc -Dextra_sections -DVAR1=5 # preprocess
Terminal> doconce format latex yourdoc extra_sections=True VAR1=5 # mako
```

The variable `FORMAT` is always defined as the current format when running `preprocess` or `mako`. That is, in the last example, `FORMAT` is defined as `latex`. Inside the Doconce document one can then perform format specific actions through tests like `#if FORMAT == "latex"` (for `preprocess`) or `% if FORMAT == "latex":` (for `mako`).

Removal of inline comments

The command-line arguments `--no-preprocess` and `--no-mako` turn off running `preprocess` and `mako`, respectively.

Inline comments in the text are removed from the output by:

```
Terminal> doconce format latex mydoc --skip_inline_comments
```

One can also remove all such comments from the original Doconce file by running:

```
Terminal> doconce remove_inline_comments mydoc
```

This action is convenient when a Doconce document reaches its final form and comments by different authors should be removed.

Notes

Doconce does not have a tag for longer notes, because implementation of a “notes feature” is so easy using the `preprocess` or `mako` programs. Just introduce some variable, say `NOTES`, that you define through `-DNOTES` (or not) when running `doconce format . . .`. Inside the document you place your notes between `# #ifdef NOTES` and `# #endif` preprocess tags. Alternatively you use `% if NOTES:` and `% endif` that `mako` will recognize. In the same way you may encapsulate unfinished material, extra material to be removed for readers but still nice to archive as part of the document for future revisions.

Demo of different formats

A simple scientific report is available in [a lot of different formats](#). How to create the different formats is explained in more depth in the coming sections.

HTML

Making an HTML version of a Doconce file `mydoc.do.txt` is performed by:

```
Terminal> doconce format html mydoc
```

The resulting file `mydoc.html` can be loaded into any web browser for viewing.

The HTML style can be defined either in the header of the HTML file, using a named built-in style; in an external CSS file; or in a template file.

An external CSS file `filename` used by setting the command-line argument `--css=filename`. There available built-in styles are specified as `--html-style=name`, where `name` can be

- `solarized`: the famous [solarized](#) style (yellowish),
- `blueish`: a simple style with blue headings (default),
- `blueish2`: a variant of *blueish*,
- `bloodish`: as *blueish*, but dark read as color.

Using `--css=filename` where `filename` is a non-existing file makes Doconce write the built-in style to that file. Otherwise the HTML links to the CSS stylesheet in `filename`. Several stylesheets can be specified: `--css=file1.css,file2.css,file3.css`.

Templates are HTML files with “slots” `%(main)s` for the main body of text, `%(title)s` for the title, and `%(date)s` for the date. Doconce comes with a few templates. The usage of templates is described in a [separate document](#). That document describes how your Doconce-generated HTML file can have any specified layout.

If the Pygments package (including the `pygmentize` program) is installed, code blocks are typeset with aid of this package. The command-line argument `--no-pygments-html` turns off the use of Pygments and makes code blocks appear with plain (pre) HTML tags. The option `--pygments-html-linenos` turns on line numbers in Pygments-formatted code blocks. A specific Pygments style is set by `--pygments-html-style=style`, where `style` can be `default`, `emacs`, `perldoc`, and other valid names for Pygments styles.

The HTML file can be embedded in a template if the Doconce document does not have a title (because then there will be no header and footer in the HTML file). The template file must contain valid HTML code and can have three “slots”: `%(title)s`

for a title, `%(date)s` for a date, and `%(main)s` for the main body of text, i.e., the Doconce document translated to HTML. The title becomes the first heading in the Doconce document, and the date is extracted from the `DATE:` line, if present. With the template feature one can easily embed the text in the look and feel of a website. The template can be extracted from the source code of a page at the site; just insert `%(title)s` and `%(date)s` at appropriate places and replace the main bod of text by `%(main)s`. Here is an example:

```
Terminal> doconce format html mydoc --html-template=mytemplate.html
```

Blogs

Doconce can be used for writing blogs provided the blog site accepts raw HTML code. Google's Blogger service (`blogger.com` or `blogname.blogspot.com`) is particularly well suited since it also allows extensive LaTeX mathematics via MathJax.

1. Write the blog text as a Doconce document without any title, author, and date.
2. Generate HTML as described above.
3. Copy the text and paste it into the text area in the blog (just delete the HTML code that initially pops up in the text area). Make sure the input format is HTML.

See a [simple blog example](#) and a [scientific report](#) for demonstrations of blogs at `blogspot.no`.

Warning

In the comments after the blog one cannot paste raw HTML code with MathJax scripts so there is no support for mathematics in the comments.

system-message

WARNING/2 in `tutorial.rst`, line 331
Duplicate explicit target name: "scientific report".

WordPress (`wordpress.com`) allows raw HTML code in blogs, but has very limited LaTeX support, basically only formulas. The `--wordpress` option to `doconce` modifies the HTML code such that all equations are typeset in a way that is acceptable to WordPress. Look at a [simple doconce example](#) and a [scientific report](#) to see blogging with mathematics and code on WordPress.

Pandoc and Markdown

Output in Pandoc's extended Markdown format results from:

```
Terminal> doconce format pandoc mydoc
```

The name of the output file is `mydoc.mkd`. From this format one can go to numerous other formats:

```
Terminal> pandoc -R -t mediawiki -o mydoc.mwk --toc mydoc.mkd
```


Pandoc supports `latex`, `html`, `odt` (OpenOffice), `docx` (Microsoft Word), `rtf`, `texinfo`, to mention some. The `-R` option makes Pandoc pass raw HTML or LaTeX to the output format instead of ignoring it, while the `--toc` option generates a table of contents. See the [Pandoc documentation](#) for the many features of the pandoc program. The HTML output from pandoc needs adjustments to provide full support for MathJax LaTeX mathematics, and for this purpose one should use `doconce md2html`:

```
Terminal> doconce format pandoc mydoc
Terminal> doconce m2html mydoc
```

The result `mydoc.html` can be viewed in a browser.

Pandoc is useful to go from LaTeX mathematics to, e.g., HTML or MS Word. There are two ways (experiment to find the best one for your document): `doconce format pandoc` and then translating using `doconce md2latex` (which runs `pandoc`), or `doconce format latex`, and then going from LaTeX to the desired format using `pandoc`. Here is an example on the latter strategy:

```
Terminal> doconce format latex mydoc
Terminal> doconce ptex2tex mydoc
Terminal> doconce replace '\Verb!' '\verb!' mydoc.tex
Terminal> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

When we go through `pandoc`, only single equations, `align`, or `align*` environments are well understood for output to HTML.

Note that `Doconce` applies the `Verb` macro from the `fancyvrb` package while `pandoc` only supports the standard `verb` construction for inline verbatim text. Moreover, quite some additional `doconce replace` and `doconce subst` edits might be needed on the `.mkd` or `.tex` files to successfully have mathematics that is well translated to MS Word. Also when going to `reStructuredText` using `Pandoc`, it can be advantageous to go via LaTeX.

Here is an example where we take a `Doconce` snippet (without title, author, and date), maybe with some unnumbered equations, and quickly generate HTML with mathematics displayed by `MathJax`:

```
Terminal> doconce format pandoc mydoc
Terminal> pandoc -t html -o mydoc.html -s --mathjax mydoc.mkd
```

The `-s` option adds a proper header and footer to the `mydoc.html` file. This recipe is a quick way of making HTML notes with (some) mathematics.

LaTeX

Making a LaTeX file `mydoc.tex` from `mydoc.do.txt` is done in two steps: ..

Note: putting code blocks inside a list is not successful in many

Step 1. Filter the `doconce` text to a pre-LaTeX form `mydoc.p.tex` for the `ptex2tex` program (or `doconce ptex2tex`):

```
Terminal> doconce format latex mydoc
```

LaTeX-specific commands (“`newcommands`”) in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see the section [Macros \(Newcommands\)](#), [Cross-References](#), [Index](#), and [Bibliography](#)).

If these files are present, they are included in the LaTeX document so that your commands are defined.

An option `--latex-printed` makes some adjustments for documents aimed at being printed. For example, links to web resources are associated with a footnote listing the complete web address (URL).

Step 2. Run `ptex2tex` (if you have it) to make a standard LaTeX file:

```
Terminal> ptex2tex mydoc
```

In case you do not have `ptex2tex`, you may run a (very) simplified version:

```
Terminal> doconce ptex2tex mydoc
```

Note that Doconce generates a `.p.tex` file with some preprocessor macros that can be used to steer certain properties of the LaTeX document. For example, to turn on the Helvetica font instead of the standard Computer Modern font, run:

```
Terminal> ptex2tex -DHELVETICA mydoc
Terminal> doconce ptex2tex mydoc -DHELVETICA # alternative
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. However, the standard LaTeX “maketitle” heading is also available through `-DLATEX_HEADING=traditional`. A separate titlepage can be generated by `-DLATEX_HEADING=titlepage`.

Preprocessor variables to be defined or undefined are

- `BOOK` for the “book” documentclass rather than the standard “article” class (necessary if you apply chapter headings with 9 =)
- `PALATINO` for the Palatino font
- `HELVETICA` for the Helvetica font
- `A4PAPER` for A4 paper size
- `A6PAPER` for A6 paper size (suitable for reading PDFs on phones)
- `MOVIE15` for using the movie15 LaTeX package to display movies
- `PREAMBLE` to turn the LaTeX preamble on or off (i.e., complete document or document to be included elsewhere - and note that the preamble is only included if the document has a title, author, and date)
- `MINTED` for inclusion of the minted package for typesetting of code with the Pygments tool (which requires `latex` or `pdflatex` to be run with the `-shell-escape` option)

If you are not satisfied with the Doconce preamble, you can provide your own preamble by adding the command-line option `--latex-preamble=myfile`. In case `myfile` contains a documentclass definition, Doconce assumes that the file contains the *complete* preamble you want (not that all the packages listed in the default preamble are required and must be present in `myfile`). Otherwise, `myfile` is assumed to contain *additional* LaTeX code to be added to the Doconce default preamble.

The `ptex2tex` tool makes it possible to easily switch between many different fancy formatings of computer or verbatim code in LaTeX documents. After any `!bc` command in the Doconce source you can insert verbatim block styles as defined in your

.ptex2tex.cfg file, e.g., !bc sys for a terminal session, where sys is set to a certain environment in .ptex2tex.cfg (e.g., CodeTerminal). There are about 40 styles to choose from, and you can easily add new ones.

Also the doconce ptex2tex command supports preprocessor directives for processing the .p.tex file. The command allows specifications of code environments as well. Here is an example:

```
Terminal> doconce ptex2tex mydoc -DLATEX_HEADING=traditional \
-DPALATINO -DA6PAPER \
"sys=\begin{quote}\begin{verbatim}@\\end{verbatim}\\end{quote}" \
fpro=minted fcod=minted shcod=Verbatim envir=ans:nt
```

Note that @ must be used to separate the begin and end LaTeX commands, unless only the environment name is given (such as minted above, which implies \begin{minted}{fortran} and \end{minted} as begin and end for blocks inside !bc fpro and !ec). Specifying envir=ans:nt means that all other environments are typeset with the anslistings.sty package, e.g., !bc cppcod will then result in \begin{c++}. If no environments like sys, fpro, or the common envir are defined on the command line, the plain \begin{verbatim} and \end{verbatim} used.

Step 2b (optional). Edit the mydoc.tex file to your needs. For example, you may want to substitute section by section* to avoid numbering of sections, you may want to insert linebreaks (and perhaps space) in the title, etc. This can be automatically edited with the aid of the doconce replace and doconce subst commands. The former works with substituting text directly, while the latter performs substitutions using regular expressions. You will use doconce replace to edit section{ to section*{:

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
```

For fixing the line break of a title, you may pick a word in the title, say “Using”, and insert a break after than word. With doconce subst this is easy employing regular expressions with a group before “Using” and a group after:

```
Terminal> doconce subst 'title\{(.+)Using (.+)\}' \
'title{\g<1> \\\ [1.5mm] Using \g<2>}' mydoc.tex
```

A lot of tailored fixes to the LaTeX document can be done by an appropriate set of text replacements and regular expression substitutions. You are anyway encouraged to make a script for generating PDF from the LaTeX file so the doconce subst or doconce replace commands can be put inside the script.

Step 3. Compile mydoc.tex and create the PDF file:

```
Terminal> latex mydoc
Terminal> latex mydoc
Terminal> makeindex mydoc    # if index
Terminal> bibitem mydoc      # if bibliography
Terminal> latex mydoc
Terminal> dvipdf mydoc
```

If one wishes to run ptex2tex and use the minted LaTeX package for typesetting code blocks (Minted_Python, Minted_Cpp, etc., in ptex2tex specified

through the `*pro` and `*cod` variables in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the minted LaTeX package is needed. This package is included by running `ptex2tex` with the `-DMINTED` option:

```
Terminal> ptex2tex -DMINTED mydoc
```

In this case, `latex` must be run with the `-shell-escape` option:

```
Terminal> latex -shell-escape mydoc
Terminal> latex -shell-escape mydoc
Terminal> makeindex mydoc      # if index
Terminal> bibitem mydoc        # if bibliography
Terminal> latex -shell-escape mydoc
Terminal> dvipdf mydoc
```

When running `doconce ptex2tex mydoc envir=minted` (or other minted specifications with `doconce ptex2tex`), the minted package is automatically included so there is no need for the `-DMINTED` option.

PDFLaTeX

Running `pdflatex` instead of `latex` follows almost the same steps, but the start is:

```
Terminal> doconce format latex mydoc
```

Then `ptex2tex` is run as explained above, and finally:

```
Terminal> pdflatex -shell-escape mydoc
Terminal> makeindex mydoc      # if index
Terminal> bibitem mydoc        # if bibliography
Terminal> pdflatex -shell-escape mydoc
```

Plain ASCII Text

We can go from Doconce “back to” plain untagged text suitable for viewing in terminal windows, inclusion in email text, or for insertion in computer source code:

```
Terminal> doconce format plain mydoc.do.txt # results in mydoc.txt
```

reStructuredText

Going from Doconce to `reStructuredText` gives a lot of possibilities to go to other formats. First we filter the Doconce text to a `reStructuredText` file `mydoc.rst`:

```
Terminal> doconce format rst mydoc.do.txt
```

We may now produce various other formats:

```
Terminal> rst2html.py mydoc.rst > mydoc.html # html
Terminal> rst2latex.py mydoc.rst > mydoc.tex  # latex
Terminal> rst2xml.py  mydoc.rst > mydoc.xml   # XML
Terminal> rst2odt.py  mydoc.rst > mydoc.odt   # OpenOffice
```

The OpenOffice file `mydoc.odt` can be loaded into OpenOffice and saved in, among other things, the RTF format or the Microsoft Word format. However, it is more convenient to use the program `unovonv` to convert between the many formats OpenOffice supports *on the command line*. Run:

```
Terminal> unoconv --show
```

to see all the formats that are supported. For example, the following commands take `mydoc.odt` to Microsoft Office Open XML format, classic MS Word format, and PDF:

```
Terminal> unoconv -f ooxml mydoc.odt
Terminal> unoconv -f doc mydoc.odt
Terminal> unoconv -f pdf mydoc.odt
```

Remark about Mathematical Typesetting. At the time of this writing, there is no easy way to go from Doconce and LaTeX mathematics to reST and further to OpenOffice and the “MS Word world”. Mathematics is only fully supported by `latex` as output and to a wide extent also supported by the `sphinx` output format. Some links for going from LaTeX to Word are listed below.

- <http://ubuntuforums.org/showthread.php?t=1033441>
- <http://tug.org/utilities/texconv/textopc.html>
- <http://nileshbansal.blogspot.com/2007/12/latex-to-openofficeword.html>

Sphinx

Sphinx documents demand quite some steps in their creation. We have automated most of the steps through the `doconce sphinx_dir` command:

```
Terminal> doconce sphinx_dir author="authors' names" \
          title="some title" version=1.0 dirname=sphinxdir \
          theme=mytheme file1 file2 file3 ...
```

The keywords `author`, `title`, and `version` are used in the headings of the Sphinx document. By default, `version` is 1.0 and the script will try to deduce authors and title from the doconce files `file1`, `file2`, etc. that together represent the whole document. Note that none of the individual Doconce files `file1`, `file2`, etc. should include the rest as their union makes up the whole document. The default value of `dirname` is `sphinx-rootdir`. The `theme` keyword is used to set the theme for design of HTML output from Sphinx (the default theme is ‘default’).

With a single-file document in `mydoc.do.txt` one often just runs:

```
Terminal> doconce sphinx_dir mydoc
```

and then an appropriate Sphinx directory `sphinx-rootdir` is made with relevant files.

The `doconce sphinx_dir` command generates a script `automake_sphinx.py` for compiling the Sphinx document into an HTML document. One can either run `automake_sphinx.py` or perform the steps in the script manually, possibly with necessary modifications. Normally, executing the script works well, but if you are new

to Sphinx and end up producing quite some Sphinx documents, I encourage you to read the Sphinx documentation and study the `automake_sphinx.py` file.

Links. The `automake_sphinx.py` script copies directories named `fig*` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. It also examines `MOVIE:` and `FIGURE:` commands in the Doconce file to find other image files and copies these too. I strongly recommend to put files to which there are local links (not `http:` or `file:` URLs) in a directory named `_static`. The `automake_sphinx.py` copies `_static*` to the Sphinx directory, which guarantees that the links to the local files will work in the Sphinx document.

There is a utility `doconce sphinxfix_localURLs` for checking links to local files and moving the files to `_static` and changing the links accordingly. For example, a link to `dir1/dir2/myfile.txt` is changed to `_static/myfile.txt` and `myfile.txt` is copied to `_static`. However, I recommend instead that you manually copy files to `_static` when you want to link to them, or let your script which compiles the Doconce document do it automatically.

Themes. Doconce comes with a rich collection of HTML themes for Sphinx documents, much larger than what is found in the standard Sphinx distribution. Additional themes include `agni`, `basicstrap`, `bootstrap`, `cloud`, `fenics`, `fenics_minimal`, `flask`, `haiku`, `impressjs`, `jal`, `pylons`, `redcloud`, `scipy_lectures`, `slim-agogo`, and `vlinux-theme`.

All the themes are packed out in the Sphinx directory, and the `doconce sphinx_dir` insert lots of extra code in the `conf.py` file to enable easy specification and customization of themes. For example, modules are loaded for the additional themes that come with Doconce, code is inserted to allow customization of the look and feel of themes, etc. The `conf.py` file is a good starting point for fine-tuning your favorite team, and your own `conf.py` file can later be supplied and used when running `doconce sphinx_dir`: simply add the command-line option `conf.py=conf.py`.

A script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics`, `pyramid`, and `pylon` one writes:

```
Terminal> ./make-themes.sh fenics pyramid pylon
```

The resulting directories with HTML documents are `_build/html_fenics` and `_build/html_pyramid`, respectively. Without arguments, `make-themes.sh` makes all available themes (!). With `make-themes.sh` it is easy to check out various themes to find the one that is most attractive for your document.

You may supply your own theme and avoid copying all the themes that come with Doconce into the Sphinx directory. Just specify `theme_dir=path` on the command line, where `path` is the relative path to the directory containing the Sphinx theme. You must also specify a configure file by `conf.py=path`, where `path` is the relative path to your `conf.py` file.

Example. Say you like the `scipy_lectures` theme, but you want a table of contents to appear *to the right*, much in the same style as in the default theme (where the table of contents is to the left). You can then run `doconce sphinx_dir`, invoke a text editor with the `conf.py` file, find the line `html_theme == 'scipy_lectures'`, edit the following `nosidebar` to `false` and `rightsidebar` to `true`. Alternatively, you may write a little script using `doconce replace` to replace a portion of text in `conf.py` by a new one:

```
doconce replace "elif html_theme == 'scipy_lectures':"
```

```

html_theme_options = {
    'nosidebar': 'true',
    'rightsidebar': 'false',
    'sidebarbgcolor': '#f2f2f2',
    'sidebartextcolor': '#20435c',
    'sidebarlinkcolor': '#20435c',
    'footerbgcolor': '#000000',
    'relbarbgcolor': '#000000',
}
elif html_theme == 'scipy_lectures':
html_theme_options = {
    'nosidebar': 'false',
    'rightsidebar': 'true',
    'sidebarbgcolor': '#f2f2f2',
    'sidebartextcolor': '#20435c',
    'sidebarlinkcolor': '#20435c',
    'footerbgcolor': '#000000',
    'relbarbgcolor': '#000000',
}
conf.py

```

Obviously, we could also have changed colors in the edit above. The final alternative is to save the edited `conf.py` file somewhere and reuse it the next time `doconce sphinx_dir` is run:

```

doconce sphinx_dir theme=scipy_lectures \
    conf.py=../some/path/conf.py mydoc

```

The manual Sphinx procedure

If it is not desirable to use the autogenerated scripts explained above, here is the complete manual procedure of generating a Sphinx document from a file `mydoc.do.txt`.

Step 1. Translate Doconce into the Sphinx format:

```
Terminal> doconce format sphinx mydoc
```

Step 2. Create a Sphinx root directory either manually or by using the interactive `sphinx-quickstart` program. Here is a scripted version of the steps with the latter:

```

mkdir sphinx-rootdir
sphinx-quickstart <<EOF
sphinx-rootdir
n
-
Name of My Sphinx Document
Author
version
version
.rst
index
n
y
n

```

```
n
n
n
Y
n
n
Y
Y
Y
EOF
```

The autogenerated `conf.py` file may need some edits if you want to specific layout (Sphinx themes) of HTML pages. The `doconce sphinx_dir` generator makes an extended `conv.py` file where, among other things, several useful Sphinx extensions are included.

Step 3. Copy the `mydoc.rst` file to the Sphinx root directory:

```
Terminal> cp mydoc.rst sphinx-rootdir
```

If you have figures in your document, the relative paths to those will be invalid when you work with `mydoc.rst` in the `sphinx-rootdir` directory. Either edit `mydoc.rst` so that figure file paths are correct, or simply copy your figure directories to `sphinx-rootdir`. Links to local files in `mydoc.rst` must be modified to links to files in the `_static` directory, see comment above.

Step 4. Edit the generated `index.rst` file so that `mydoc.rst` is included, i.e., add `mydoc` to the `toctree` section so that it becomes:

```
.. toctree::
   :maxdepth: 2

   mydoc
```

(The spaces before `mydoc` are important!)

Step 5. Generate, for instance, an HTML version of the Sphinx source:

```
make clean    # remove old versions
make html
```

Sphinx can generate a range of different formats: standalone HTML, HTML in separate directories with `index.html` files, a large single HTML file, JSON files, various help files (the `qthelp`, `HTML`, and `Devhelp` projects), `epub`, `LaTeX`, `PDF` (via `LaTeX`), pure text, man pages, and `Texinfo` files.

Step 6. View the result:

```
Terminal> firefox _build/html/index.html
```

Note that verbatim code blocks can be typeset in a variety of ways depending the argument that follows `!bc`: `cod` gives Python (code-block:: python in Sphinx syntax) and `cppcod` gives C++, but all such arguments can be customized both for Sphinx and `LaTeX` output.

Wiki Formats

There are many different wiki formats, but Doconce only supports three: [Googlecode wiki](#), [MediaWiki](#), and [Creole Wiki](#). These formats are called `gwiki`, `mwiki`, and `cwiki`, respectively. Transformation from Doconce to these formats is done by:

```
Terminal> doconce format gwiki mydoc.do.txt
Terminal> doconce format mwiki mydoc.do.txt
Terminal> doconce format cwiki mydoc.do.txt
```

The produced MediaWiki can be tested in the [sandbox of wikibooks.org](#). The format works well with Wikipedia, Wikibooks, and [ShoutWiki](#), but not always well elsewhere (see [this example](#)).

Large MediaWiki documents can be made with the [Book creator](#). From the MediaWiki format one can go to other formats with aid of [mwlib](#). This means that one can easily use Doconce to write [Wikibooks](#) and publish these in PDF and MediaWiki format, while at the same time, the book can also be published as a standard LaTeX book, a Sphinx web document, or a collection of HTML files.

The Googlecode wiki document, `mydoc.gwiki`, is most conveniently stored in a directory which is a clone of the wiki part of the Googlecode project. This is far easier than copying and pasting the entire text into the wiki editor in a web browser.

When the Doconce file contains figures, each figure filename must in the `.gwiki` file be replaced by a URL where the figure is available. There are instructions in the file for doing this. Usually, one performs this substitution automatically (see next section).

Tweaking the Doconce Output

Occasionally, one would like to tweak the output in a certain format from Doconce. One example is figure filenames when transforming Doconce to reStructuredText. Since Doconce does not know if the `.rst` file is going to be filtered to LaTeX or HTML, it cannot know if `.eps` or `.png` is the most appropriate image filename. The solution is to use a text substitution command or code with, e.g., `sed`, `perl`, `python`, or `scitools subst`, to automatically edit the output file from Doconce. It is then wise to run Doconce and the editing commands from a script to automate all steps in going from Doconce to the final format(s). The `make.sh` files in `docs/manual` and `docs/tutorial` constitute comprehensive examples on how such scripts can be made.

Demos

The current text is generated from a Doconce format stored in the file:

```
docs/tutorial/tutorial.do.txt
```

The file `make.sh` in the `tutorial` directory of the Doconce source code contains a demo of how to produce a variety of formats. The source of this tutorial, `tutorial.do.txt` is the starting point. Running `make.sh` and studying the various generated files and comparing them with the original `tutorial.do.txt` file, gives a quick introduction to how Doconce is used in a real case. [Here](#) is a sample of how this tutorial looks in different formats.

There is another demo in the `docs/manual` directory which translates the more comprehensive documentation, `manual.do.txt`, to various formats. The `make.sh` script runs a set of translations.

Installation of Doconce and its Dependencies

Doconce

Doconce itself is pure Python code hosted at <http://code.google.com/p/doconce>. Its installation from the Mercurial (hg) source follows the standard procedure:

```
# Doconce
hg clone https://code.google.com/p/doconce/ doconce
cd doconce
sudo python setup.py install
cd ..
```

Since Doconce is frequently updated, it is recommended to use the above procedure and whenever a problem occurs, make sure to update to the most recent version:

```
cd doconce
hg pull
hg update
sudo python setup.py install
```

Debian GNU/Linux users can also run:

```
sudo apt-get install doconce
```

This installs the latest release and not the most updated and bugfixed version. On Ubuntu one needs to run:

```
sudo add-apt-repository ppa:scitools/ppa
sudo apt-get update
sudo apt-get install doconce
```

Dependencies

Preprocessors

If you make use of the [Preprocess](#) preprocessor, this program must be installed:

```
svn checkout http://preprocess.googlecode.com/svn/trunk/ preprocess
cd preprocess
cd doconce
sudo python setup.py install
cd ..
```

A much more advanced alternative to Preprocess is [Mako](#). Its installation is most conveniently done by pip:

```
pip install Mako
```

This command requires pip to be installed. On Debian Linux systems, such as Ubuntu, the installation is simply done by:

```
sudo apt-get install python-pip
```

Alternatively, one can install from the pip [source code](#).

Mako can also be installed directly from [source](#): download the tarball, pack it out, go to the directory and run the usual `sudo python setup.py install`.

Image file handling

Different output formats require different formats of image files. For example, PostScript or Encapsulated PostScript is required for `latex` output, while HTML needs JPEG, GIF, or PNG formats. Doconce calls up programs from the ImageMagick suite for converting image files to a proper format if needed. The [ImageMagick suite](#) can be installed on all major platforms. On Debian Linux (including Ubuntu) systems one can simply write:

```
sudo apt-get install imagemagick
```

The convenience program `doconce combine_images`, for combining several images into one, will use `montage` and `convert` from ImageMagick and the `pdftk`, `pdfnup`, and `pdfcrop` programs from the `texlive-extra-utils` Debian package. The latter gets installed by:

```
sudo apt-get install texlive-extra-utils
```

Spellcheck

The utility `doconce spellcheck` applies the `ispell` program for spellcheck. On Debian (including Ubuntu) it is installed by:

```
sudo apt-get install ispell
```

Ptex2tex for LaTeX Output

To make LaTeX documents with very flexible choice of typesetting of verbatim code blocks you need [ptex2tex](#), which is installed by:

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
```

It may happen that you need additional style files, you can run a script, `cp2texmf.sh`:

```
cd latex
sh cp2texmf.sh # copy stylefiles to ~/texmf directory
cd ../..
```

This script copies some special stylefiles that that `ptex2tex` potentially makes use of. Some more standard stylefiles are also needed. These are installed by:

```
sudo apt-get install texlive-latex-recommended texlive-latex-extra
```

on Debian Linux (including Ubuntu) systems. TeXShop on Mac comes with the necessary stylefiles (if not, they can be found by googling and installed manually in the `~/texmf/tex/latex/misc` directory).

Note that the `doconce ptex2tex` command, which needs no installation beyond Doconce itself, can be used as a simpler alternative to the `ptex2tex` program.

The *minted* LaTeX style is offered by `ptex2tex` and `doconce ptext2tex` and popular among many users. This style requires the package [Pygments](#) to be installed. On Debian Linux:

```
sudo apt-get install python-pygments
```

Alternatively, the package can be installed manually:

```
hg clone ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments
cd pygments
sudo python setup.py install
```

If you use the minted style together with `ptex2tex`, you have to enable it by the `-DMINTED` command-line argument to `ptex2tex`. This is not necessary if you run the alternative `doconce ptex2tex` program.

All use of the minted style requires the `-shell-escape` command-line argument when running LaTeX, i.e., `latex -shell-escape` or `pdflatex -shell-escape`.

reStructuredText (reST) Output

The `rst` output from Doconce allows further transformation to LaTeX, HTML, XML, OpenOffice, and so on, through the [docutils](#) package. The installation of the most recent version can be done by:

```
svn checkout http://docutils.svn.sourceforge.net/svnroot/docutils/trunk/docutils
cd docutils
sudo python setup.py install
cd ..
```

To use the OpenOffice suite you will typically on Debian systems install:

```
sudo apt-get install unovonv libreoffice libreoffice-dmaths
```

There is a possibility to create PDF files from reST documents using ReportLab instead of LaTeX. The enabling software is [rst2pdf](#). Either download the tarball or clone the svn repository, go to the `rst2pdf` directory and run the usual `sudo python setup.py install`.

Output to sphinx requires of course the [Sphinx software](#), installed by:

```
hg clone https://bitbucket.org/birkenfeld/sphinx
cd sphinx
sudo python setup.py install
cd ..
```

Markdown and Pandoc Output

The Doconce format `pandoc` outputs the document in the Pandoc extended Markdown format, which via the `pandoc` program can be translated to a range of other formats. Installation of [Pandoc](#), written in Haskell, is most easily done by:

```
sudo apt-get install pandoc
```

on Debian (Ubuntu) systems.

Epydoc Output

When the output format is epydoc one needs that program too, installed by:

```
svn co https://epydock.svn.sourceforge.net/svnroot/epydock/trunk/epydock epydock
cd epydock
sudo make install
cd ..
```

Remark. Several of the packages above installed from source code are also available in Debian-based system through the `apt-get install` command. However, we recommend installation directly from the version control system repository as there might be important updates and bug fixes. For svn directories, go to the directory, run `svn update`, and then `sudo python setup.py install`. For Mercurial (hg) directories, go to the directory, run `hg pull; hg update`, and then `sudo python setup.py install`.