

” **tutorial.do.txt** ”

Missing: FIGURE, MOVIE, environments

TITLE: Doconce: Document Once, Include Anywhere

AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of Oslo

DATE: today

* 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": "<http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf>", "HTML": "<http://www.htmlcodetutorial.com/>", "reStructuredText": "<http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html>", "Sphinx": "<http://sphinx.pocoo.org/contents.html>", and "wiki": "<http://code.google.com/p/support/wiki/WikiSyntax>"? 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,

” **tutorial.do.txt** ”

- * 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:

```
!bc
===== 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}`.

tutorial.do.txt

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

!ec

The Doconce text above results in the following little document:

==== 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 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:

- 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 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 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

tutorial.do.txt

4.0	1.1E+1	14.717624
-----	--------	-----------

==== Mathematics and Computer Code ====

Inline mathematics, such as $\nu = \sin(x)$ or $v = \sin(x)$, 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 $\nu = \sin(x)$ or $v = \sin(x)$ is typeset as

```
!bc
 $\nu = \sin(x)$   $v = \sin(x)$ 
!ec
```

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

```
!bc
 $\nu = \sin(x)$ 
!ec
```

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:

```
!bc
|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
!ec
```

Note: |bt and |et (and |bc and |ec below) are used to illustrate
tex and code blocks in inside verbatim blocks and are replaced
by !bt, !et, !bc, and !ec after all other formatting is finished.
The result looks like this:

```
!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
```

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.

” **tutorial.do.txt** ”

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

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

```
|ec
!ec
```

Such blocks are formatted as

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

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

```
!ec
```

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 ====
label{newcommands}
```

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).

” **tutorial.do.txt** ”

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": "<https://doconce.googlecode.com/hg/doc/demos/manual/index.html>" for various formats of this document).

Example on including another Doconce file (using preprocess):

#include "_doconce2anything.do.txt"

===== Demos =====

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

!bc

docs/tutorial/tutorial.do.txt

!ec

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": "<https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html>" 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.

#include "../manual/install.do.txt"

Doconce: Document Once, Include Anywhere

Hans Petter Langtangen^{1,2}

¹Simula Research Laboratory

²University of Oslo

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, L^AT_EX, 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.

1 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:

1.1 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 Section 1.1.

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 Section 2 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

1.2 Mathematics and Computer Code

Inline mathematics, such as $\nu = \sin(x)$, 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 $\nu = \sin(x)$ is typeset as

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

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:

$$\frac{\partial u}{\partial t} = \nabla^2 u + f, \tag{1}$$

$$\frac{\partial v}{\partial t} = \nabla \cdot (q(u) \nabla v) + g \tag{2}$$

Of course, such blocks only look 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 give you plain text editing capabilities.

1.3 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).

2 From Doconce to Other Formats

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

Terminal

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

or just

Terminal

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

2.1 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

Terminal

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

to produce the necessary output for generating HTML, PDF \LaTeX , 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.

2.2 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

```
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`).

2.3 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

```
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.

2.4 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.

2.5 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.

2.6 HTML

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

Terminal

```
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 red 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: `--ccs=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


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

2.7 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.

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.

2.8 Pandoc and Markdown

Output in Pandoc's extended Markdown format results from

Terminal

```
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

```
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

```
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

```
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

```
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.

2.9 L^AT_EX

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

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

Terminal

```
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 Section 1.3). If these files are present, they are included in the L^AT_EX 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 L^AT_EX file,

Terminal

```
Terminal> ptex2tex mydoc
```

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

Terminal

```
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 L^AT_EX document. For example, to turn on the Helvetica font instead of the standard Computer Modern font, run

Terminal

```
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 formattings 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

```
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 L^AT_EX commands, unless only the environment name is given (such as `\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

```
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

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

A lot of tailored fixes to the L^AT_EX 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 L^AT_EX 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

```
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 L^AT_EX 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

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

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

Terminal

```
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.

2.10 PDFLaTeX

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

Terminal

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

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

Terminal

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

2.11 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

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

2.12 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

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

We may now produce various other formats:

Terminal

```
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 `unoconv` to convert between the many formats OpenOffice supports *on the command line*. Run

Terminal

```
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

```
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>

2.13 Sphinx

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

Terminal

```
Terminal> doconce sphinx_dir author="authors' names" \  
          title="some title" version=1.0 dirname=sphinx_dir \  
          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

```
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

```
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',
        'sidebargbgcolor': '#f2f2f2',
        'sidebarbgcolor': '#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

```

Terminal
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
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:

```

Terminal
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

```
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:

Terminal

```
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

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

Note that verbatim code blocks can be typeset in a variety of ways depending on the argument that follows `bcl:` `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.

2.14 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

```
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).

2.15 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.

2.16 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.

3 Installation of Doconce and its Dependencies

3.1 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:

Terminal

```
# 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:

Terminal

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

Debian GNU/Linux users can also run

Terminal

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

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

Terminal

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

3.2 Dependencies

Preprocessors. If you make use of the Preprocess preprocessor, this program must be installed:

Terminal

```
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,

Terminal

```
pip install Mako
```

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

Terminal

```
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

Terminal

```
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

Terminal

```
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

Terminal

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

Ptex2tex for L^AT_EX Output. To make L^AT_EX documents with very flexible choice of typesetting of verbatim code blocks you need `ptex2tex`, which is installed by

Terminal

```
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`:

Terminal

```
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

Terminal

```
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,

Terminal

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

Alternatively, the package can be installed manually:

Terminal

```
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

Terminal

```
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

Terminal

```
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

Terminal

```
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

Terminal

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

on Debian (Ubuntu) systems.

Epydoc Output. When the output format is epydoc one needs that program too, installed by

Terminal

```
svn co https://epydock.svn.sourceforge.net/svnroot/epydock/trunk/epydock epydoc
cd epydoc
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`.

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`.

Doconce: Document Once, Include Anywhere Documentation

Release 1.0

Hans Petter Langtangen

March 05, 2013

CONTENTS

1	Doconce: Document Once, Include Anywhere	3
2	What Does Doconce Look Like?	5
2.1	A Subsection with Sample Text	6
2.2	Mathematics and Computer Code	7
2.3	Macros (Newcommands), Cross-References, Index, and Bibliography	8
3	From Doconce to Other Formats	9
3.1	Generating a makefile	9
3.2	Preprocessing	9
3.3	Removal of inline comments	10
3.4	Notes	10
3.5	Demo of different formats	10
3.6	HTML	10
3.7	Blogs	11
3.8	Pandoc and Markdown	11
3.9	LaTeX	12
3.10	PDFLaTeX	14
3.11	Plain ASCII Text	15
3.12	reStructuredText	15
3.13	Sphinx	15
3.14	Wiki Formats	18
3.15	Tweaking the Doconce Output	19
3.16	Demos	19
4	Installation of Doconce and its Dependencies	21
4.1	Doconce	21
4.2	Dependencies	21
5	Indices and tables	25

Contents:

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 |
|---r---r-----r---|
| 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:

2.1 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

2.2 Mathematics and Computer Code

Inline mathematics, such as $\nu = \sin(x)$, 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 $\nu = \sin(x)$ is typeset as

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

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:

$$\frac{\partial u}{\partial t} = \nabla^2 u + f, \tag{2.1}$$

$$\frac{\partial v}{\partial t} = \nabla \cdot (q(u) \nabla v) + g$$

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.

2.3 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
```

3.1 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.

3.2 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`).

3.3 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.

3.4 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.

3.5 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.

3.6 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 red 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
```

3.7 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.

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.

3.8 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.

3.9 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 formattings 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.

3.10 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
```

3.11 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
```

3.12 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 `unoconv` 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>

3.13 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=sphinx_dir \
          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
```

3.13.1 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 `conf.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.

3.14 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).

3.15 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.

3.16 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

4.1 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
```

4.2 Dependencies

4.2.1 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.
```

4.2.2 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
```

4.2.3 Spellcheck

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

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

4.2.4 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`.

4.2.5 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 ..
```

4.2.6 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.

4.2.7 Epydoc Output

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

```
svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
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`.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

” **tutorial.txt** ”

Doconce: Document Once, Include Anywhere

=====

Hans Petter Langtangen [1, 2]

[1] Simula Research Laboratory

[2] University of Oslo

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 (<http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf>), HTML (<http://www.htmlcodetutorial.com/>), reStructuredText (<http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html>), Sphinx (<http://sphinx.pocoo.org/contents.html>), and wiki (<http://code.google.com/p/support/wiki/WikiSyntax>)? 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

” **tutorial.txt** ”

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`

” .

” **tutorial.txt** ”

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` (`http://folk.uio.no/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

” **tutorial.txt** ”

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::

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

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

tutorial.txt

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

” **tutorial.txt** ”

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 (<https://doconce.googlecode.com/hg/doc/demos/manual/index.html>) 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.

tutorial.txt

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      # pre
process
Terminal> doconce format latex yourdoc extra_sections=True VAR1=5  # mak
o

```

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

” **tutorial.txt** ”

A simple scientific report is available in a lot of different formats (http://hplgit.github.com/teamods/writing_reports/doconce_commands.html). 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 (<http://ethanschoonover.com/solarized>) style (yellowish),
- * blueish: a simple style with blue headings (default),
- * blueish2: a variant of *blueish*,
- * bloodish: as blueish, but dark red 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 (https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html). 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

tutorial.txt

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 (<http://doconce.blogspot.no>) and a scientific report (<http://doconce-report-demo.blogspot.no/>) 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.

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 (<http://doconce.wordpress.com>) and a scientific report (<http://doconcereportdemo.wordpress.com/>) 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
```

” **tutorial.txt** ”

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 (<http://johnmacfarlane.net/pandoc/README.html>) 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 my 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

tutorial.txt

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

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 generate 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)

” **tutorial.txt** ”

- * MOVIE15 for using the moviel5 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 formattings 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 enviro=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 `enviro=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 `enviro` 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*{`:

” **tutorial.txt** ”

```
!bc sys
```

```
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

tutorial.txt

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::

” **tutorial.txt** ”

```
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=sphinx_dir \
          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

” **tutorial.txt** ”

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

” **tutorial.txt** ”

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
```

## tutorial.txt

\*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
```

” **tutorial.txt** ”

(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 (<http://code.google.com/p/support/wiki/WikiSyntax>), MediaWiki (<http://www.mediawiki.org/wiki/Help:Formatting>), and Creole Wiki (<http://www.wikicreole.org/wiki/Creole1.0>). 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 (<http://en.wikibooks.org/wiki/Sandbox>). The format works well with Wikipedia, Wikibooks, and ShoutWiki ([http://doconcedemo.shoutwiki.com/wiki/Doconce\\_demo\\_page](http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page)), but not always well elsewhere (see this example ([http://doconcedemo.jumpwiki.com/wiki/First\\_demo](http://doconcedemo.jumpwiki.com/wiki/First_demo))).

Large MediaWiki documents can be made with the Book creator ([http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book\\_creator](http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator)).

From the MediaWiki format one can go to other formats with aid of mwlib (<http://pediapress.com/code/>). This means that one can easily use Doconce to write Wikibooks (<http://en.wikibooks.org>) and publish these in PDF and MediaWiki format, while

**tutorial.txt**

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 (<https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html>) 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>.

” **tutorial.txt** ”

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 (<http://code.google.com/p/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 (<http://www.makotemplates.org>). Its installation is most conveniently done by pip::

```
pip install Mako
```

tutorial.txt

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 (<http://pypi.python.org/pypi/pip>).

Mako can also be installed directly from source (<http://www.makotemplates.org/download.html>): 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 (<http://www.imagemagick.org/script/index.php>) 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` (<http://code.google.com/p/ptex2tex>), which is installed by::

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
```

” **tutorial.txt** ”

```
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 (<http://pygments.org>) 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 (<http://docutils.sourceforge.net>) package. The installation of the most recent version can be done by::

” **tutorial.txt** ”

```

    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 (<http://code.google.com/p/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 (<http://sphinx.pocoo.org>), 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 (<http://johnmacfarlane.net/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::

```

ydoc    svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc ep
        cd epydoc
        sudo make install
        cd ..

```

”

tutorial.txt

”

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`.

tutorial.epytext

TITLE: Doconce: Document Once, Include Anywhere

BY: Hans Petter Langtangen (Simula Research Laboratory, and University of Oslo)

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 U{LaTeX<<http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf>>}, U{HTML<<http://www.htmlcodetutorial.com/>>}, U{reStructuredText<<http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html>>}, U{Sphinx<<http://sphinx.pocoo.org/contents.html>>}, and U{wiki<<http://code.google.com/p/support/wiki/WikiSyntax>>}? 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 C{*}.
- I{Emphasized words} are surrounded by C{*}.
- B{Words in boldface} are surrounded by underscores.
- Words from computer code are enclosed in back quotes and then typeset C{verbatim} (in a monospace font).
- Section headings are recognized by equality (C{=}) signs before and after the title, and the number of C{=} 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 I{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 C{!bc} (begin code) and C{!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 C{!bt} (begin TeX) and C{!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

tutorial.epytext

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

”

tutorial.epytext

”

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 `B{boldface}` words, `I{emphasized}` words, and `C{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 `C{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 `U{hpl<http://folk.uio.no/hpl>}`. If the word is URL, the URL itself becomes the link name, as in `U{tutorial.do.txt<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 `M{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 `M{v = sin(x)}` is typeset as::

NOTE: A verbatim block has been removed because it causes problems for Epytext.

”

tutorial.epytext

”

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 `C{latex}`, `C{pdflatex}`, `C{html}`, `C{sphinx}`, and `C{pandoc}` are of interest and all these support inline LaTeX mathematics so then you will naturally drop the pipe symbol and write just::

NOTE: A verbatim block has been removed because it causes problems for Epytext.

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 `C{!bt}` and `C{!et}` (begin TeX, end TeX) instructions::

NOTE: A verbatim block has been removed because it causes problems for Epytext.

The result looks like this::

NOTE: A verbatim block has been removed because it causes problems for Epytext.

Of course, such blocks only looks nice in formats with support for LaTeX mathematics, and here the `align` environment in particular (this includes `C{latex}`, `C{pdflatex}`, `C{html}`, and `C{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 `C{!bc}` and `C{!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)
```

”

”

”

tutorial.epytext

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

A code block must come after some plain sentence (at least for successful output to C{sphinx}, C{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 C{# #include "mynote.do.txt"} at the beginning of a line. Doconce documents have extension C{do.txt}. The C{do} part stands for doconce, while the trailing C{.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 I{newcommand} construction. The newcommands defined in a file with name C{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 C{newcommands.tex} and C{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 C{!bt} and C{!et} in C{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 C{newcommands_replace.tex} and expanded by Doconce. The definitions of newcommands in the C{newcommands*.tex} files I{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 C{doc/manual/manual.do.txt} file (see the U{demo page<<https://doconce.googlecode.com/hg/doc/demos/manual/index.html>>} for various formats of this document).

From Doconce to Other Formats

tutorial.epytext

=====

Transformation of a Doconce document C{mydoc.do.txt} to various other formats applies the script C{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 C{doconce makefile} can be used to automatically generate such a makefile, more precisely a Bash script C{make.sh}, which carries out the commands explained below. If our Doconce source is in C{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 C{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 C{make.sh} script is that it inserts checks for successful runs of the C{doconce format} commands, and if something goes wrong, the C{make.sh} exits.

Preprocessing

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

```
Terminal> doconce format latex mydoc -Dextra_sections -DVAR1=5      # pre
process
Terminal> doconce format latex yourdoc extra_sections=True VAR1=5  # mak
o
```

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

Removal of inline comments

” **tutorial.epytext** ”

The command-line arguments `C{--no-preprocess}` and `C{--no-mako}` turn off running `C{preprocess}` and `C{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 `C{preprocess}` or `C{mako}` programs. Just introduce some variable, say `C{NOTES}`, that you define through `C{-DNOTES}` (or not) when running `C{doconce format ...}`. Inside the document you place your notes between `C{# #ifdef NOTES}` and `C{# #endif}` preprocess tags. Alternatively you use `C{% if NOTES:}` and `C{% endif}` that `C{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 U{a lot of different formatshttp://hplgit.github.com/teamods/writing_reports/doconce_commands.html}. How to create the different formats is explained in more depth in the coming sections.

HTML

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

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

The resulting file `C{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 `C{filename}` used by setting the command-line argument `C{--css=filename}`. There available built-in styles are specified as `C{--html-style=name}`, where `C{name}` can be

tutorial.epytext

- C{solarized}: the famous U{solarized<<http://ethanschoonover.com/solarized>>} style (yellowish),
- C{blueish}: a simple style with blue headings (default),
- C{blueish2}: a variant of I{bluish},
- C{bloodish}: as C{bluish}, but dark read as color.

Using C{--css=filename} where C{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 C{filename}. Several stylesheets can be specified: C{--css=file1.css,file2.css,file3.css}.

Templates are HTML files with "slots" C{%(main)s} for the main body of text, C{%(title)s} for the title, and C{%(date)s} for the date. Doconce comes with a few templates. The usage of templates is described in a U{separate document<https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html>}. That document describes how you your Doconce-generated HTML file can have any specified layout.

If the Pygments package (including the C{pygmentize} program) is installed, code blocks are typeset with aid of this package. The command-line argument C{--no-pygments-html} turns off the use of Pygments and makes code blocks appear with plain (C{pre}) HTML tags. The option C{--pygments-html-linenos} turns on line numbers in Pygments-formatted code blocks. A specific Pygments style is set by C{--pygments-html-style=style}, where C{style} can be C{default}, C{emacs}, C{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": C{%(title)s} for a title, C{%(date)s} for a date, and C{%(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 C{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 C{%(title)s} and C{%(date)s} at appropriate places and replace the main bod of text by C{%(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 (C{blogger.com} or C{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

” **tutorial.epytext** ”

pops up in the text area). Make sure the input format is HTML.

See a U{simple blog example<<http://doconce.blogspot.no>>} and a U{scientific report<<http://doconce-report-demo.blogspot.no/>>}} for demonstrations of blogs at C{blogspot.no}.

I{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.

WordPress (C{wordpress.com}) allows raw HTML code in blogs, but has very limited LaTeX support, basically only formulas. The C{--wordpress} option to C{doconce} modifies the HTML code such that all equations are typeset in a way that is acceptable to WordPress. Look at a U{simple doconce example<<http://doconce.wordpress.com>>} and a U{scientific report<<http://doconcereportdemo.wordpress.com/>>}} 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 C{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 C{latex}, C{html}, C{odt} (OpenOffice), C{docx} (Microsoft Word), C{rtf}, C{texinfo}, to mention some. The C{-R} option makes Pandoc pass raw HTML or LaTeX to the output format instead of ignoring it, while the C{--toc} option generates a table of contents. See the U{Pandoc documentation<<http://johnmacfarlane.net/pandoc/README.html>>}} for the many features of the C{pandoc} program. The HTML output from C{pandoc} needs adjustments to provide full support for MathJax LaTeX mathematics, and for this purpose one should use C{doconce md2html}::

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

The result C{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): C{doconce format pandoc} and then translating using C{doconce md2latex} (which runs C{pandoc}), or C{doconce format latex}, and then going from LaTeX to the desired format using C{pandoc}. Here is an example on the latter strategy::

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

tutorial.epytext

```
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 `C{pandoc}`, only single equations, `C{align}`, or `C{align*}` environments are well understood for output to HTML.

Note that Doconce applies the `C{Verb}` macro from the `C{fancyvrb}` package while `C{pandoc}` only supports the standard `C{verb}` construction for inline verbatim text. Moreover, quite some additional `C{doconce replace}` and `C{doconce subst}` edits might be needed on the `C{.mkd}` or `C{.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 my MathJax::

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

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

LaTeX

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

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

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

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in files `C{newcommands.tex}`, `C{newcommands_keep.tex}`, or `C{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 `C{--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).

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

```
Terminal> ptex2tex mydoc
```

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

```
Terminal> doconce ptex2tex mydoc
```

” **tutorial.epytext** ”

Note that Doconce generates a `C{.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 `C{-DLATEX_HEADING=traditional}`. A separate titlepage can be generate by `C{-DLATEX_HEADING=titlepage}`.

Preprocessor variables to be defined or undefined are

- `C{BOOK}` for the "book" documentclass rather than the standard "article" class (necessary if you apply chapter headings with 9 `C{=}`)
- `C{PALATINO}` for the Palatino font
- `C{HELVETICA}` for the Helvetica font
- `C{A4PAPER}` for A4 paper size
- `C{A6PAPER}` for A6 paper size (suitable for reading PDFs on phones)
- `C{MOVIE15}` for using the movie15 LaTeX package to display movies
- `C{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)
- `C{MINTED}` for inclusion of the minted package for typesetting of code with the Pygments tool (which requires `C{latex}` or `C{pdflatex}` to be run with the `C{-shell-escape}` option)

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

The `C{ptex2tex}` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any `C{!bc}` command in the Doconce source you can insert verbatim block styles as defined in your `C{.ptex2tex.cfg}` file, e.g., `C{!bc sys}` for a terminal session, where `C{sys}` is set to a certain environment in `C{.ptex2tex.cfg}` (e.g., `C{CodeTerminal}`). There are about 40 styles to choose from, and you can easily add new ones.

Also the `C{doconce ptex2tex}` command supports preprocessor directives for processing the `C{.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}"
```

” **tutorial.epytext** ”

```
\
    fpro=minted fcod=minted shcod=Verbatim envir=ans:nt
```

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

I{Step 2b (optional).} Edit the `C{mydoc.tex}` file to your needs. For example, you may want to substitute `C{section}` by `C{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 `C{doconce replace}` and `C{doconce subst}` commands. The former works with substituting text directly, while the latter performs substitutions using regular expressions. You will use `C{doconce replace}` to edit `C{section{}}` to `C{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 `C{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 `C{doconce subst}` or `C{doconce replace}` commands can be put inside the script.

I{Step 3.} Compile `C{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 `C{ptex2tex}` and use the minted LaTeX package for typesetting code blocks (`C{Minted_Python}`, `C{Minted_Cpp}`, etc., in `C{ptex2tex}` specified through the `C{*pro}` and `C{*cod}` variables in `C{.ptex2tex.cfg}` or `C{$HOME/.ptex2tex.cfg}`), the minted LaTeX package is needed. This package is included by running `C{ptex2tex}` with the `C{-DMINTED}` option::

tutorial.epytext

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

In this case, `C{latex}` must be run with the `C{-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 `C{doconce ptex2tex mydoc envir=minted}` (or other minted specifications with `C{doconce ptex2tex}`), the minted package is automatically included so there is no need for the `C{-DMINTED}` option.

PDFLaTeX

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

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

Then `C{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 `C{mydoc.rst}`::

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

” **tutorial.epytext** ”

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 C{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 C{unovonv} to convert between the many formats OpenOffice supports I{on the command line}. Run::

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

to see all the formats that are supported. For example, the following commands take C{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
```

I{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 C{latex} as output and to a wide extent also supported by the C{sphinx} output format. Some links for going from LaTeX to Word are listed below.

- U{<http://ubuntuforums.org/showthread.php?t=1033441><<http://ubuntuforums.org/showthread.php?t=1033441>>}
- U{<http://tug.org/utilities/texconv/textopc.html><<http://tug.org/utilities/texconv/textopc.html>>}
- U{<http://nileshbansal.blogspot.com/2007/12/latex-to-openofficeword.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 C{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 C{author}, C{title}, and C{version} are used in the headings of the Sphinx document. By default, C{version} is 1.0 and the script will try to deduce authors and title from the doconce files C{file1}, C{file2}, etc. that together represent the whole document. Note that none of the individual Doconce files C{file1}, C{file2}, etc. should

”

tutorial.epytext

”

include the rest as their union makes up the whole document. The default value of `C{dirname}` is `C{sphinx-rootdir}`. The `C{theme}` keyword is used to set the theme for design of HTML output from Sphinx (the default theme is `C{'default'}`).

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

```
Terminal> doconce sphinx_dir mydoc
```

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

The `C{doconce sphinx_dir}` command generates a script `C{automake_sphinx.py}` for compiling the Sphinx document into an HTML document. One can either run `C{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 `C{automake_sphinx.py}` file.

I{Links.} The `C{automake_sphinx.py}` script copies directories named `C{fig*}` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. It also examines `C{MOVIE:}` and `C{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 `C{http:}` or `C{file:}` URLs) in a directory named `C{_static}`. The `C{automake_sphinx.py}` copies `C{_static*}` to the Sphinx directory, which guarantees that the links to the local files will work in the Sphinx document.

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

I{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

```
C{agni},
C{basicstrap},
C{bootstrap},
C{cloud},
C{fenics},
C{fenics_minimal},
C{flask},
C{haiku},
C{impressjs},
C{jal},
C{pylons},
C{redcloud},
C{scipy_lectures},
C{slim-agogo}, and
C{vlinux-theme}.
```

”

”

”

” **tutorial.epytext** ”

All the themes are packed out in the Sphinx directory, and the `C{doconce sphinx_dir}` insert lots of extra code in the `C{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 `C{conf.py}` file is a good starting point for fine-tuning your favorite team, and your own `C{conf.py}` file can later be supplied and used when running `C{doconce sphinx_dir}`: simply add the command-line option `C{conf.py=conf.py}`.

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

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

The resulting directories with HTML documents are `C{_build/html_fenics}` and `C{_build/html_pyramid}`, respectively. Without arguments, `C{make-themes.sh}` makes all available themes (!). With `C{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 `C{theme_dir=path}` on the command line, where `C{path}` is the relative path to the directory containing the Sphinx theme. You must also specify a configure file by `C{conf.py=path}`, where `C{path}` is the relative path to your `C{conf.py}` file.

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

”

tutorial.epytext

”

```

        '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 C{conf.py} file somewhere and reuse it the next time C{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 C{mydoc.do.txt}.

I{Step 1.} Translate Doconce into the Sphinx format::

```

Terminal> doconce format sphinx mydoc

```

I{Step 2.} Create a Sphinx root directory either manually or by using the interactive C{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 C{conf.py} file may need some edits if you want to specific layout (Sphinx themes) of HTML pages. The C{doconce sphinx\_dir} generator makes an extended C{conv.py}

”

”

”

”

**tutorial.epytext**

”

file where, among other things, several useful Sphinx extensions are included.

I{Step 3.} Copy the C{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 C{mydoc.rst} in the C{sphinx-rootdir} directory. Either edit C{mydoc.rst} so that figure file paths are correct, or simply copy your figure directories to C{sphinx-rootdir}. Links to local files in C{mydoc.rst} must be modified to links to files in the C{\_static} directory, see comment above.

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

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

```
   mydoc
```

(The spaces before C{mydoc} are important!)

I{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 C{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.

I{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 C{!bc}: C{cod} gives Python (C{code-block:: python} in Sphinx syntax) and C{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: U{Googlecode wiki<<http://code.google.com/p/support/wiki/WikiSyntax>>}, U{MediaWiki<<http://www.mediawiki.org/wiki/Help:Formatting>>}, and U{Creole Wiki<<http://www.wikicreole.org/wiki/Creole1.0>>}.

”

”

”

”

**tutorial.epytext**

”

These formats are called C{gwiki}, C{mwiki}, and C{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 U{sandbox of wikibooks.org<<http://en.wikibooks.org/wiki/Sandbox>>}. The format works well with Wikipedia, Wikibooks, and U{ShoutWiki<[http://doconcedemo.shoutwiki.com/wiki/Doconce\\_demo\\_page](http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page)>}, but not always well elsewhere (see U{this example<[http://doconcedemo.jumpwiki.com/wiki/First\\_demo](http://doconcedemo.jumpwiki.com/wiki/First_demo)>}).

Large MediaWiki documents can be made with the U{Book creator<[http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book\\_creator](http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator)>}. From the MediaWiki format one can go to other formats with aid

of U{mwlib<<http://pediapress.com/code/>>}. This means that one can easily use Doconce to write U{Wikibooks<<http://en.wikibooks.org>>} 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, C{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 C{.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 C{.rst} file is going to be filtered to LaTeX or HTML, it cannot know if C{.eps} or C{.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 C{make.sh} files in C{docs/manual} and C{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::

”

”

”

” **tutorial.epytext** ”

docs/tutorial/tutorial.do.txt

The file C{make.sh} in the C{tutorial} directory of the Doconce source code contains a demo of how to produce a variety of formats. The source of this tutorial, C{tutorial.do.txt} is the starting point. Running C{make.sh} and studying the various generated files and comparing them with the original C{tutorial.do.txt} file, gives a quick introduction to how Doconce is used in a real case. U{Here<<https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html>>} is a sample of how this tutorial looks in different formats.

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

#### Installation of Doconce and its Dependencies

=====

#### Doconce

-----

Doconce itself is pure Python code hosted at U{<http://code.google.com/p/doconce><<http://code.google.com/p/doconce>>}. Its installation from the Mercurial (C{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
```

## tutorial.epytext

### Dependencies

### Preprocessors

If you make use of the U{Preprocess<<http://code.google.com/p/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 U{Mako<<http://www.makotemplates.org>>}. Its installation is most conveniently done by C{pip}::

```
pip install Mako
```

This command requires C{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 C{pip} U{source code<<http://pypi.python.org/pypi/pip>>}.

Mako can also be installed directly from U{source<<http://www.makotemplates.org/download.html>>}: download the tarball, pack it out, go to the directory and run the usual C{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 C{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 U{ImageMagick suite<<http://www.imagemagick.org/script/index.php>>} 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 C{doconce combine\_images}, for combining several images into one, will use C{montage} and C{convert} from ImageMagick and the C{pdftk}, C{pdfnup}, and C{pdfcrop} programs from the C{texlive-extra-utils} Debian package. The latter gets installed by::

” **tutorial.epytext** ”

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

### Spellcheck ~~~~~

The utility `C{doconce spellcheck}` applies the `C{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 `U{ptex2tex<http://code.google.com/p/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, `C{cp2texmf.sh}`::

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

This script copies some special stylefiles that that `C{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 `C{~/texmf/tex/latex/misc}` directory).

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

The `I{minted}` LaTeX style is offered by `C{ptex2tex}` and `C{doconce ptext2tex}` and popular among many users. This style requires the package `U{Pygments<http://pygments.org>}` to be installed. On Debian Linux::

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

Alternatively, the package can be installed manually::



”

**tutorial.epytext**

”

```
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 `C{ptex2tex}`, you have to enable it by the `C{-DMINTED}` command-line argument to `C{ptex2tex}`. This is not necessary if you run the alternative `C{doconce ptex2tex}` program.

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

#### reStructuredText (reST) Output

~~~~~

The `C{rst}` output from Doconce allows further transformation to LaTeX, HTML, XML, OpenOffice, and so on, through the `U{docutils<http://docutils.sourceforge.net>}` 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 `U{rst2pdf<http://code.google.com/p/rst2pdf>}`. Either download the tarball or clone the svn repository, go to the `C{rst2pdf}` directory and run the usual `C{sudo python setup.py install}`.

Output to `C{sphinx}` requires of course the `U{Sphinx software<http://sphinx.pocoo.org>}`, installed by::

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

Markdown and Pandoc Output

~~~~~

The Doconce format `C{pandoc}` outputs the document in the Pandoc

”

”

”

”

**tutorial.epytext**

”

extended Markdown format, which via the C{pandoc} program can be translated to a range of other formats. Installation of U{Pandoc<<http://johnmacfarlane.net/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 C{epydok} one needs that program too, installed by::

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

I{Remark.} Several of the packages above installed from source code are also available in Debian-based system through the C{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 C{svn} directories, go to the directory, run C{svn update}, and then C{sudo python setup.py install}. For Mercurial (C{hg}) directories, go to the directory, run C{hg pull; hg update}, and then C{sudo python setup.py install}.

tutorial.gwiki

<wiki:comment> Missing: FIGURE, MOVIE, environments </wiki:comment>

#summary Doconce: Document Once, Include Anywhere

By *Hans Petter Langtangen*

==== 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 [<http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf> LaTeX], [<http://www.htmlcodetutorial.com/HTML>], [<http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html> reStructuredText], [<http://sphinx.pocoo.org/contents.html> Sphinx], and [<http://code.google.com/p/support/wiki/WikiSyntax> 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.

tutorial.gwiki

* 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 =====
```

tutorial.gwiki

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:

```
# item 1
# item 2
# item 3
```

URLs with a link word are possible, as in `[http://folk.uio.no/hpl 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.,

	<i>*time*</i>		<i>*velocity*</i>		<i>*acceleration*</i>	
	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

```
{{ {{
```

tutorial.gwiki

```

 $\nu = \sin(x)$ 

```

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
}}}

```

`<wiki:comment>` Note: `!bt` and `!et` (and `!bc` and `!ec` below) are used to illustrate `</wiki:comment>`

`<wiki:comment>` tex and code blocks in inside verbatim blocks and are replaced `</wiki:comment>`

`<wiki:comment>` by `!bt`, `!et`, `!bc`, and `!ec` after all other formatting is finished. `</wiki:comment>`

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)

```

tutorial.gwiki

```
}}}
```

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 [https://doconce.googlecode.com/hg/doc/demos/manual/index.html demo page] for various formats of this document).

```
<wiki:comment> Example on including another Doconce file (using preprocess): </wiki:comment>
```

== From Doconce to Other Formats ==

tutorial.gwiki

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:

tutorial.gwiki

```

{{{
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 [http://hplgit.github.com/teamods/writing_reports/doconce_commands.html 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 [<http://ethanschoonover.com/solarized>] style (yellowish),
- * 'blueish': a simple style with blue headings (default),
- * 'blueish2': a variant of *blueish*,
- * 'bloodish': as 'blueish', but dark red 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 [https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html]

tutorial.gwiki

eparate document]. That document describes how you 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.

```

# Write the blog text as a Doconce document without any title, author, and date.
# Generate HTML as described above.
# 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 [<http://doconce.blogspot.no> simple blog example] and a [<http://doconce-report-demo.blogspot.no/> 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.

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.

tutorial.gwiki

Look at a [<http://doconce.wordpress.com> simple doconce example] and a [<http://doconcereportdemo.wordpress.com/> 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 [<http://johnmacfarlane.net/pandoc/README.html> 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 my MathJax:

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

” **tutorial.gwiki** ”

```
}}}
```

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:

```
<wiki:comment> Note: putting code blocks inside a list is not successful in many
</wiki:comment>
```

```
<wiki:comment> formats - the text may be messed up. A better choice is a paragraph
</wiki:comment>
```

```
<wiki:comment> environment, as used here. </wiki:comment>
```

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 -DHELIVETICA mydoc
```

```
Terminal> doconce ptex2tex mydoc -DHELIVETICA # 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

tutorial.gwiki

```
* '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 includ
ed if the document has a title, author, and date)
* 'MINTED' for inclusion of the minted package for typesetting of code with t
he Pygments tool (which requires 'latex' or 'pdflatex' to be run with the '-sh
ell-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 formattings 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
}
```

”

tutorial.gwiki

”

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
}}}
```

”

”

”

”

tutorial.gwiki

”

```
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>

tutorial.gwiki

==== 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=sphinx_dir \
           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'`,

tutorial.gwiki

```
`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',
```

tutorial.gwiki

```

        '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

```

tutorial.gwiki

```

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 ====

tutorial.gwiki

There are many different wiki formats, but Doconce only supports three:
[\[http://code.google.com/p/support/wiki/WikiSyntax Googlecode wiki\]](http://code.google.com/p/support/wiki/WikiSyntax),
[\[http://www.mediawiki.org/wiki/Help:Formatting MediaWiki\]](http://www.mediawiki.org/wiki/Help:Formatting), and
[\[http://www.wikicreole.org/wiki/Creole1.0 Creole Wiki\]](http://www.wikicreole.org/wiki/Creole1.0).

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 [\[http://en.wikibooks.org/wiki/Sandbox\]](http://en.wikibooks.org/wiki/Sandbox) sandbox of

wikibooks.org]. The format

works well with Wikipedia, Wikibooks, and

[\[http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page ShoutWiki\]](http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page),

but not always well elsewhere

(see [\[http://doconcedemo.jumpwiki.com/wiki/First_demo\]](http://doconcedemo.jumpwiki.com/wiki/First_demo) this example)).

Large MediaWiki documents can be made with the

[\[http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator\]](http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator) Book creator].

From the MediaWiki format one can go to other formats with aid

of [\[http://pediapress.com/code/mwlib\]](http://pediapress.com/code/mwlib). This means that one can

easily use Doconce to write [\[http://en.wikibooks.org Wikibooks\]](http://en.wikibooks.org)

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 ====

”

tutorial.gwiki

”

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. [<https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html> 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 ====

”

”

”

tutorial.gwiki

If you make use of the [<http://code.google.com/p/preprocess> 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 [<http://www.makotemplates.org> 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' [<http://pypi.python.org/pypi/pip> source code].

Mako can also be installed directly from [<http://www.makotemplates.org/download.html> 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 [<http://www.imagemagick.org/script/index.php> 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

tutorial.gwiki

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 [<http://code.google.com/p/ptex2tex> 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 [<http://pygments.org> 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

”

tutorial.gwiki

”

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

`<wiki:comment>` Say something about `anslistings.sty` `</wiki:comment>`

==== reStructuredText (reST) Output ====

The `'rst'` output from Doconce allows further transformation to LaTeX, HTML, XML, OpenOffice, and so on, through the [<http://docutils.sourceforge.net> 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 [<http://code.google.com/p/rst2pdf> 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 [<http://sphinx.pocoo.org> 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 [<http://johnmacfarlane.net/pandoc/> 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

”

”

”

”

tutorial.gwiki

”

```
{{{
svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
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'`.

tutorial.md

```
<!-- Missing: FIGURE, MOVIE, environments -->
```

```
% Doconce: Document Once, Include Anywhere
```

```
% Hans Petter Langtangen at Simula Research Laboratory and University of Oslo
```

```
% 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](http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf), [HTML](http://www.htmlcodetutorial.com/), [reStructuredText](http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html), [Sphinx](http://sphinx.pocoo.org/contents.html), and [wiki](http://code.google.com/p/support/wiki/WikiSyntax)? 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

tutorial.md

`'!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}`.

tutorial.md

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](http://folk.uio.no/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](#t).

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](#s) for an example).

Tables are also supported, e.g.,

time	velocity	acceleration
------	----------	--------------

tutorial.md

```

-----
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 $\nu = \sin(x)$, 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 $\nu = \sin(x)$ is typeset as

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

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, \label{myeq1} \\
\{\partial v \over \partial t\} &= \nabla \cdot (q(u) \nabla v) + g \\
\end{align}
!et
~~~~~
```

```
<!-- Note: !bt and !et (and !bc and !ec below) are used to illustrate -->
<!-- tex and code blocks in inside verbatim blocks and are replaced -->
<!-- by !bt, !et, !bc, and !ec after all other formatting is finished. -->
The result looks like this:
```

```
$$
\begin{equation}
\{\partial u \over \partial t\} = \nabla^2 u + f, \label{myeq1}
\end{equation}
$$
```

tutorial.md

```


$$\frac{\partial v}{\partial t} = \nabla \cdot (q(u) \nabla v) + g$$


```

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

```

~~~~~{.Python}
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*`

tutorial.md

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](<https://doconce.googlecode.com/hg/doc/demos/manual/index.html>) for various formats of this document).

```
<!-- Example on including another Doconce file (using preprocess): -->
```

From Doconce to Other Formats

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

```
~~~~~{.Bash}
Terminal> doconce format format mydoc.do.txt
~~~~~
```

or just

```
~~~~~{.Bash}
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

tutorial.md

carries out the commands explained below. If our Doconce source is in 'main_myproj.do.txt', we run

```
~~~~~{.Bash}
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,

```
~~~~~{.Bash}
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

```
~~~~~{.Bash}
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'

tutorial.md

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](http://hplgit.github.com/teamods/writing_reports/doconce_commands.html). 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

```
~~~~~{.Bash}
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](<http://ethanschoonover.com/solarized>) style (yellowish),
- * `'blueish'`: a simple style with blue headings (default),
- * `'blueish2'`: a variant of `*blueish*`,
- * `'bloodish'`: as `'blueish'`, but dark red 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](https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html). 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'`

tutorial.md

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:

```
~~~~~{.Bash}
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](<http://doconce.blogspot.no>) and a [scientific report](<http://doconce-report-demo.blogspot.no/>) 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.

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](<http://doconce.wordpress.com>) and a [scientific report](<http://doconcereportdemo.wordpress.com/>) to see blogging with mathematics and code on WordPress.

Pandoc and Markdown

tutorial.md

Output in Pandoc's extended Markdown format results from

```
~~~~~{.Bash}
Terminal> doconce format pandoc mydoc
~~~~~
```

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

```
~~~~~{.Bash}
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](<http://johnmacfarlane.net/pandoc/README.html>) 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':

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

tutorial.md

```
~~~~~{.Bash}
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 -->`
`<!-- formats - the text may be messed up. A better choice is a paragraph -->`
`<!-- environment, as used here. -->`

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

```
~~~~~{.Bash}
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](#y)). 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,

```
~~~~~{.Bash}
Terminal> ptex2tex mydoc
~~~~~
```

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

```
~~~~~{.Bash}
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

```
~~~~~{.Bash}
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

tutorial.md

institutions in common. However, the standard LaTeX "maketitle" heading is also available through `'-DLATEX_HEADING=traditional'`.

A separate titlepage can be generate 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 formattings 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:

```
~~~~~{.Bash}
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'`)

” **tutorial.md** ”

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*{`:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
```

tutorial.md

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

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

```
~~~~~{.Bash}
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

```
~~~~~{.Bash}
Terminal> doconce format latex mydoc
~~~~~
```

Then 'ptex2tex' is run as explained above, and finally

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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':

```
~~~~~{.Bash}
Terminal> doconce format rst mydoc.do.txt
~~~~~
```

We may now produce various other formats:

tutorial.md

```
~~~~~{.Bash}
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

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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.

tutorial.md

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

```
~~~~~{.Bash}
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'.
```

tutorial.md

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

```
~~~~~{.Bash}
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',
~~~~~
```

tutorial.md

```
'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

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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
~~~~~
```

” **tutorial.md** ”

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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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:

```
~~~~~{.Bash}
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

tutorial.md

There are many different wiki formats, but Doconce only supports three: [Googlecode wiki](http://code.google.com/p/support/wiki/WikiSyntax), [MediaWiki](http://www.mediawiki.org/wiki/Help:Formatting), and [Creole Wiki](http://www.wikicreole.org/wiki/Creole1.0).

These formats are called

'gwiki', 'mwiki', and 'cwiki', respectively.

Transformation from Doconce to these formats is done by

```
~~~~~{.Bash}
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](http://en.wikibooks.org/wiki/Sandbox). The format works well with Wikipedia, Wikibooks, and [ShoutWiki](http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page), but not always well elsewhere (see [this example](http://doconcedemo.jumpwiki.com/wiki/First_demo)).

Large MediaWiki documents can be made with the [Book creator](http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator).

From the MediaWiki format one can go to other formats with aid of [mwlib](http://pediapress.com/code/). This means that one can easily use Doconce to write [Wikibooks](http://en.wikibooks.org) 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.

tutorial.md

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](<https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html>) 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:

```
~~~~~{.Bash}
# 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:

```
~~~~~{.Bash}
cd doconce
hg pull
hg update
sudo python setup.py install
~~~~~
```

Debian GNU/Linux users can also run

```
~~~~~{.Bash}
sudo apt-get install doconce
~~~~~
```

This installs the latest release and not the most updated and bugfixed version.

On Ubuntu one needs to run

```
~~~~~{.Bash}
```

tutorial.md

```
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](http://code.google.com/p/preprocess) preprocessor, this program must be installed:

```
~~~~~{.Bash}
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](http://www.makotemplates.org). Its installation is most conveniently done by 'pip',

```
~~~~~{.Bash}
pip install Mako
~~~~~
```

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

```
~~~~~{.Bash}
sudo apt-get install python-pip
~~~~~
```

Alternatively, one can install from the 'pip' [source code](http://pypi.python.org/pypi/pip).

Mako can also be installed directly from [source](http://www.makotemplates.org/download.html): 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](http://www.imagemagick.org/script/index.php) can be installed on all major platforms. On Debian Linux (including Ubuntu) systems one can simply write

```
~~~~~{.Bash}
sudo apt-get install imagemagick
~~~~~
```

tutorial.md

```

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

~~~~~{.Bash}
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

~~~~~{.Bash}
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](http://code.google.com/p/ptex2tex),
which is installed by

~~~~~{.Bash}
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':

~~~~~{.Bash}
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

~~~~~{.Bash}
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

```


tutorial.md

beyond Doconce itself, can be used as a simpler alternative to the 'ptex2tex' program.

The **minted** LaTeX style is offered by 'ptex2tex' and 'doconce ptex2tex' and popular among many users. This style requires the package [Pygments](<http://pygments.org>) to be installed. On Debian Linux,

```
~~~~~{.Bash}
sudo apt-get install python-pygments
~~~~~
```

Alternatively, the package can be installed manually:

```
~~~~~{.Bash}
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'.

<!-- Say something about anslistings.sty -->

reStructuredText (reST) Output

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

```
~~~~~{.Bash}
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

```
~~~~~{.Bash}
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](<http://code.google.com/p/rst2pdf>). Either download the tarball or clone the svn repository, go to the 'rst2pdf' directory and run the usual 'sudo python setup.py install'.

tutorial.md

Output to 'sphinx' requires of course the [Sphinx software](<http://sphinx.pocoo.org>), installed by

```
~~~~~{.Bash}
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](<http://johnmacfarlane.net/pandoc/>), written in Haskell, is most easily done by

```
~~~~~{.Bash}
sudo apt-get install pandoc
~~~~~
```

on Debian (Ubuntu) systems.

Epydoc Output

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

```
~~~~~{.Bash}
svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
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'.

Doconce: Document Once, Include Anywhere Documentation

Release 1.0

Hans Petter Langtangen

March 05, 2013

CONTENTS

1	Doconce: Document Once, Include Anywhere	3
2	What Does Doconce Look Like?	5
2.1	A Subsection with Sample Text	6
2.2	Mathematics and Computer Code	7
2.3	Macros (Newcommands), Cross-References, Index, and Bibliography	8
3	From Doconce to Other Formats	9
3.1	Generating a makefile	9
3.2	Preprocessing	9
3.3	Removal of inline comments	10
3.4	Notes	10
3.5	Demo of different formats	10
3.6	HTML	10
3.7	Blogs	11
3.8	Pandoc and Markdown	11
3.9	LaTeX	12
3.10	PDFLaTeX	14
3.11	Plain ASCII Text	15
3.12	reStructuredText	15
3.13	Sphinx	15
3.14	Wiki Formats	18
3.15	Tweaking the Doconce Output	19
3.16	Demos	19
4	Installation of Doconce and its Dependencies	21
4.1	Doconce	21
4.2	Dependencies	21
5	Indices and tables	25

Contents:

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 |
|---r-----r-----r-----|
| 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:

2.1 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

2.2 Mathematics and Computer Code

Inline mathematics, such as $\nu = \sin(x)$, 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 $\nu = \sin(x)$ is typeset as

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

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:

$$\frac{\partial u}{\partial t} = \nabla^2 u + f, \tag{2.1}$$

$$\frac{\partial v}{\partial t} = \nabla \cdot (q(u) \nabla v) + g$$

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.

2.3 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
```

3.1 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.

3.2 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`).

3.3 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.

3.4 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.

3.5 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.

3.6 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 red 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
```

3.7 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.

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.

3.8 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.

3.9 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 formattings 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.

3.10 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
```

3.11 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
```

3.12 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 `unoconv` 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>

3.13 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=sphinx_dir \
          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
```

3.13.1 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 `conf.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.

3.14 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).

3.15 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.

3.16 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

4.1 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
```

4.2 Dependencies

4.2.1 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.
```

4.2.2 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
```

4.2.3 Spellcheck

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

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

4.2.4 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`.

4.2.5 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 ..
```

4.2.6 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.

4.2.7 Epydoc Output

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

```
svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
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`.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

tutorial.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE document PUBLIC "-//IDN docutils.sourceforge.net//DTD Docutils Generic
//EN//XML" "http://docutils.sourceforge.net/docs/ref/docutils.dtd">
<!-- Generated by Docutils 0.9 -->
<document source="tutorial.rst"><comment xml:space="preserve">Automatically gene
rated reST file from Doconce source
(http://code.google.com/p/doconce/)</comment><comment xml:space="preserve">Missi
ng: FIGURE, MOVIE, environments</comment><section ids="doconce-document-once-inc
lude-anywhere" names="doconce:\ document\ once,\ include\ anywhere"><title>Docon
ce: Document Once, Include Anywhere</title><field_list><field><field_name>Author
</field_name><field_body><paragraph>Hans Petter Langtangen</paragraph></field_bo
dy></field><field><field_name>Date</field_name><field_body><paragraph>Mar 5, 201
3</paragraph><bullet_list bullet="*"><list_item><paragraph>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?</paragraph></list_item><list_item><parag
raph>Do you need to write documents in varying formats but find it
difficult to remember all the typesetting details of various
formats like <reference name="LaTeX" refuri="http://refcards.com/docs/silvermanj
/amslatex/LaTeXRefCard.v2.0.pdf">LaTeX</reference><target ids="latex" names="lat
ex" refuri="http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCard.v2.0.pdf"/
>, <reference name="HTML" refuri="http://www.htmlcodetutorial.com/">HTML</refere
nce><target ids="html" names="html" refuri="http://www.htmlcodetutorial.com/">,
<reference name="reStructuredText" refuri="http://docutils.sourceforge.net/docs
/ref/rst/restructuredtext.html">reStructuredText</reference><target ids="restruc
turedtext" names="restructuredtext" refuri="http://docutils.sourceforge.net/docs
/ref/rst/restructuredtext.html"/>, <reference name="Sphinx" refuri="http://sphin
x.pocoo.org/contents.html">Sphinx</reference><target ids="sphinx" names="sphinx"
refuri="http://sphinx.pocoo.org/contents.html"/>, and <reference name="wiki" re
furi="http://code.google.com/p/support/wiki/WikiSyntax">wiki</reference><target
ids="wiki" names="wiki" refuri="http://code.google.com/p/support/wiki/WikiSyntax
"/>? Would it be convenient
to generate the typesetting details of a particular format from a
very simple text-like format with minimal tagging?</paragraph></list_item><list_
item><paragraph>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?</paragraph></list_item></bullet_list></field_body></field><
/field_list><paragraph>If any of these questions are of interest, you should kee
p on reading.</paragraph></section><section ids="what-does-doconce-look-like" na
mes="what\ does\ doconce\ look\ like?"><title>What Does Doconce Look Like?</titl
e><paragraph>Doconce text looks like ordinary text, but there are some almost in
visible
text constructions that allow you to control the formatting. Here are
some examples.</paragraph><block_quote><bullet_list bullet="*"><list_item><paragr
aph>Bullet lists arise from lines starting with <literal>*</literal>.</paragr
aph></list_item><list_item><paragraph><emphasis>Emphasized words</emphasis> are sur
rounded by <literal>*</literal>.</paragraph></list_item><list_item><paragraph><s
trong>Words in boldface</strong> are surrounded by underscores.</paragraph></lis
t_item><list_item><paragraph>Words from computer code are enclosed in back quote
s and
then typeset <literal>verbatim (in a monospace font)</literal>.</paragraph></lis
t_item><list_item><paragraph>Section headings are recognized by equality (<litera
l>=</literal>) signs before
and after the title, and the number of <literal>=</literal> signs indicates the

```


tutorial.xml

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 `<literal>!bc</literal>` (begin code) and `<literal>!ec</literal>` (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 `<literal>!bt</literal>` (begin TeX) and `<literal>!et</literal>` (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

tutorial.xml

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:

```
<target refid="my-first-sec"/><section ids="a-subsection-with-sample-text my-first-sec" names="a\ subsection\ with\ sample\ text my:first:sec"><title>A Subsection with Sample Text</title><paragraph>Ordinary text looks like ordinary text, and the tags used for
```

```
<strong>boldface</strong> words, <emphasis>emphasized</emphasis> words, and <literal>computer</literal> words look natural in plain text. Lists are typeset as you would do in an email,</paragrap
h><block_quote><bullet_list bullet="*"><list_item><paragraph>item 1</paragraph></list_item><list_item><paragraph>item 2</paragraph></list_item><list_item><parag
raph>item 3</paragraph></list_item></bullet_list></block_quote><paragraph>Lists can also have numbered items instead of bullets, just use an <literal>o</literal>
```

```
> (for ordered) instead of the asterisk:</paragraph><block_quote><enumerated_list enumtype="arabic" prefix="" suffix="."><list_item><paragraph>item 1</paragraph></list_item><list_item><paragraph>item 2</paragraph></list_item><list_item><parag
raph>item 3</paragraph></list_item></enumerated_list></block_quote><paragraph>URLs with a link word are possible, as in <reference name="hpl" refuri="http://folk.uio.no/hpl">hpl</reference><target ids="hpl" names="hpl" refuri="http://folk.uio.no/hpl"/>.
```

If the word is URL, the URL itself becomes the link name, as in <reference name="tutorial.do.txt" refuri="tutorial.do.txt">tutorial.do.txt</reference><target ids="tutorial-do-txt" names="tutorial.do.txt" refuri="tutorial.do.txt"/>.</paragraph><paragraph>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 <reference name="A Subsection with Sample Text" refid="a-subsection-with-sample-text">A Subsection with Sample Text</reference>.</paragraph><paragraph>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 <reference name="From Doconce to Other Formats" refid="from-doc-once-to-other-formats">From Doconce to Other Formats</reference> for an example)
.</paragraph><paragraph>Tables are also supported, e.g.,</paragraph><table><tgrou
p cols="3"><colspec colwidth="12"/><colspec colwidth="12"/><colspec colwidth="12"/><thead><row><entry><paragraph>time</paragraph></entry><entry><paragraph>velo
city</paragraph></entry><entry><paragraph>acceleration</paragraph></entry></row></thead><tbody><row><entry><paragraph>0.0</paragraph></entry><entry><paragraph>1
.4186</paragraph></entry><entry><paragraph>-5.01</paragraph></entry></row><row><
```

tutorial.xml

```

entry><paragraph>2.0</paragraph></entry><entry><paragraph>1.376512</paragraph></entry><entry><paragraph>11.919</paragraph></entry></row><row><entry><paragraph>4.0</paragraph></entry><entry><paragraph>1.1E+1</paragraph></entry><entry><paragraph>14.717624</paragraph></entry></row></tbody></tgroup></table></section><section ids="mathematics-and-computer-code" names="mathematics\ and\ computer\ code">
<title>Mathematics and Computer Code</title><paragraph>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:</paragraph><literal_block xml:space="preserve">\nu = \sin(x)$|v =
\sin(x)$</literal_block><paragraph>The pipe symbol acts as a delimiter between La
TeX code and the plain text
version of the formula. If you write a lot of mathematics, only the
output formats <literal>latex</literal>, <literal>pdflatex</literal>, <literal>h
tml</literal>, <literal>sphinx</literal>, and <literal>pandoc</literal>
are of interest
and all these support inline LaTeX mathematics so then you will naturally
drop the pipe symbol and write just:</paragraph><literal_block xml:space="preser
ve">\nu = \sin(x)$</literal_block><paragraph>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.</parag
raph><paragraph>Blocks of mathematics are typeset with raw LaTeX, inside
<literal>!bt</literal> and <literal>!et</literal> (begin TeX, end TeX) instructi
ons:</paragraph><literal_block xml:space="preserve">!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</literal_block><comment xml:space="preserve">Note: !bt and !et (and !bc and
!ec below) are used to illustrate</comment><comment xml:space="preserve">tex and
code blocks in inside verbatim blocks and are replaced</comment><comment xml:sp
ace="preserve">by !bt, !et, !bc, and !ec after all other formatting is finished.
</comment><paragraph>The result looks like this:</paragraph><literal_block xml:s
pace="preserve">\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}</literal_block><paragraph>Of course, such blocks only looks nice in
formats with support
for LaTeX mathematics, and here the align environment in particular
(this includes <literal>latex</literal>, <literal>pdflatex</literal>, <literal>h
tml</literal>, and <literal>sphinx</literal>). The raw
LaTeX syntax appears in simpler formats, but can still be useful
for those who can read LaTeX syntax.</paragraph><paragraph>You can have blocks o
f computer code, starting and ending with
<literal>!bc</literal> and <literal>!ec</literal> instructions, respectively:</p
aragraph><literal_block xml:space="preserve">!bc pycod
from math import sin, pi
def myfunc(x):
    return sin(pi*x)

import integrate
I = integrate.trapezoidal(myfunc, 0, pi, 100)
!ec</literal_block><paragraph>Such blocks are formatted as:</paragraph><literal_
block xml:space="preserve">from math import sin, pi
def myfunc(x):
    return sin(pi*x)

```

tutorial.xml

```
import integrate
I = integrate.trapezoidal(myfunc, 0, pi, 100)</literal_block><paragraph>A code b
lock must come after some plain sentence (at least for successful
output to <literal>sphinx</literal>, <literal>rst</literal>, and ASCII-close for
mats),
not directly after a section/paragraph heading or a table.</paragraph><paragraph>
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!).</paragraph><paragraph>Another document can be in
cluded by writing <literal># #include &quot;mynote.do.txt&quot;</literal>
at the beginning of a line. Doconce documents have
extension <literal>do.txt</literal>. The <literal>do</literal> part stands for d
oconce, while the
trailing <literal>.txt</literal> denotes a text document so that editors gives y
ou
plain text editing capabilities.</paragraph><target refid="newcommands"/></secti
on><section ids="macros-newcommands-cross-references-index-and-bibliography newc
ommands" names="macros\ (newcommands),\ cross-references,\ index,\ and\ bibliogr
aphy newcommands"><title>Macros (Newcommands), Cross-References, Index, and Bibl
iography</title><paragraph>Doconce supports a type of macros via a LaTeX-style <
emphasis>newcommand</emphasis>
construction. The newcommands defined in a file with name
<literal>newcommand_replace.tex</literal> are expanded when Doconce is filtered
to
other formats, except for LaTeX (since LaTeX performs the expansion
itself). Newcommands in files with names <literal>newcommands.tex</literal> and
<literal>newcommands_keep.tex</literal> 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
<literal>!bt</literal> and <literal>!et</literal> in <literal>newcommands_keep.t
ex</literal> 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
<literal>newcommands_replace.tex</literal> and expanded by Doconce. The definit
ions of
newcommands in the <literal>newcommands*.tex</literal> files <emphasis>must</emp
hasis> appear on a single
line (multi-line newcommands are too hard to parse with regular
expressions).</paragraph><paragraph>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 <literal>doc/manual/manual.do.txt</literal> file (see the
<reference name="demo page" refuri="https://doconce.googlecode.com/hg/doc/demos/
manual/index.html">demo page</reference><target ids="demo-page" names="demo\ pag
e" refuri="https://doconce.googlecode.com/hg/doc/demos/manual/index.html"/>
for various formats of this document).</paragraph><comment xml:space="preserve">
Example on including another Doconce file (using preprocess):</comment><target r
```

tutorial.xml

```

efid="doconce2formats"/></section></section><section ids="from-doconce-to-other-formats doconce2formats" names="from\ doconce\ to\ other\ formats doconce2formats"><title>From Doconce to Other Formats</title><paragraph>Transformation of a Doconce document <literal>mydoc.do.txt</literal> to various other formats applies the script <literal>doconce format</literal>:</paragraph><literal_block xml:space="preserve">Terminal> doconce format format mydoc.do.txt</literal_block><paragraph>or just:</paragraph><literal_block xml:space="preserve">Terminal> doconce format format mydoc</literal_block><section ids="generating-a-makefile" names="generating\ a\ makefile"><title>Generating a makefile</title><paragraph>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 &quot;makefile&quot;.</paragraph><paragraph>The <literal>doconce makefile</literal> can be used to automatically generate such a makefile, more precisely a Bash script <literal>make.sh</literal>, which carries out the commands explained below. If our Doconce source is in <literal>main_myproj.do.txt</literal>, we run:</paragraph><literal_block xml:space="preserve">doconce makefile main_myproj html pdflatex sphinx</literal_block><paragraph>to produce the necessary output for generating HTML, pdfLaTeX, and Sphinx. Usually, you need to edit <literal>make.sh</literal> to really fit your needs. Some example lines are inserted as comments to show various options that can be added to the basic commands. A handy feature of the generated <literal>make.sh</literal> script is that it inserts checks for successful runs of the <literal>doconce format</literal> commands, and if something goes wrong, the <literal>make.sh</literal> exits.</paragraph></section><section ids="preprocessing" names="preprocessing"><title>Preprocessing</title><paragraph>The <literal>preprocess</literal> and <literal>mako</literal> programs are used to preprocess the file, and options to <literal>preprocess</literal> and/or <literal>mako</literal> can be added after the filename. For example:</paragraph><literal_block xml:space="preserve">Terminal> doconce format latex mydoc -Dextra_sections -DVAR1=5 # preprocess
Terminal> doconce format latex yourdoc extra_sections=True VAR1=5 # mako</literal_block><paragraph>The variable <literal>FORMAT</literal> is always defined as the current format when running <literal>preprocess</literal> or <literal>mako</literal>. That is, in the last example, <literal>FORMAT</literal> is defined as <literal>latex</literal>. Inside the Doconce document one can then perform format specific actions through tests like <literal>#if FORMAT == &quot;latex&quot;ot;</literal> (for <literal>preprocess</literal>) or <literal>% if FORMAT == &quot;latex&quot;:</literal> (for <literal>mako</literal>).</paragraph></section><section ids="removal-of-inline-comments" names="removal\ of\ inline\ comments"><title>Removal of inline comments</title><paragraph>The command-line arguments <literal>--no-preprocess</literal> and <literal>--no-mako</literal> turn off running <literal>preprocess</literal> and <literal>mako</literal>, respectively.</paragraph><paragraph>Inline comments in the text are removed from the output by:</paragraph><literal_block xml:space="preserve">Terminal> doconce format latex mydoc --skip_inline_comments</literal_block><paragraph>One can also remove all such comments from the original Doconce file by running:</paragraph><literal_block xml:space="preserve">Terminal> doconce remove_inline_comments mydoc</literal_block><paragraph>This action is convenient when a Doconce document reaches its final form and comments by different authors should be removed.</paragraph></section><section ids="notes" names="notes"><title>Notes</title><paragraph>Doconce does not hav

```

tutorial.xml

e a tag for longer notes, because implementation of a "notes feature" is so easy using the `<literal>preprocess</literal>` or `<literal>mako</literal>` programs. Just introduce some variable, say `<literal>NOTES</literal>`, that you define through `<literal>-DNOTES</literal>` (or not) when running `<literal>doconce format ...</literal>`. Inside the document you place your notes between `<literal># #ifdef NOTES</literal>` and `<literal># #endif</literal>` preprocess tags. Alternatively you use `<literal>% if NOTES:</literal>` and `<literal>% endif</literal>` that `<literal>mako</literal>` 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.

`</paragraph></section><section ids="demo-of-different-formats" names="demo\ of\ different\ formats"><title>Demo of different formats</title><paragraph>A simple scientific report is available in <reference name="a lot of different formats" refuri="http://hplgit.github.com/teamods/writing_reports/doconce_commands.html">a lot of different formats</reference><target ids="a-lot-of-different-formats" names="a\ lot\ of\ different\ formats" refuri="http://hplgit.github.com/teamods/writing_reports/doconce_commands.html"/>. How to create the different formats is explained in more depth in the coming sections.`

`</paragraph></section><section dupnames="html" ids="id1"><title>HTML</title><paragraph>Making an HTML version of a Doconce file <literal>mydoc.do.txt</literal> is performed by:`

`<literal_block xml:space="preserve">Terminal> doc once format html mydoc</literal_block><paragraph>The resulting file <literal>mydoc.html</literal> can be loaded into any web browser for viewing.`

`</paragraph><paragraph>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.`

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

- `<literal>solarized</literal>`: the famous `<reference name="solarized" refuri="http://ethanschoonover.com/solarized">solarized</reference><target ids="solarized" names="solarized" refuri="http://ethanschoonover.com/solarized"/>` style (yellowish),
- `<literal>blueish</literal>`: a simple style with blue headings (default),
- `<literal>blueish2</literal>`: a variant of `<emphasis>bluish</emphasis>`,
- `<literal>bloodish</literal>`: as `<literal>bluish</literal>`, but dark read as color.

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

Templates are HTML files with "slots" `<literal>%(main)s</literal>` for the main body of text, `<literal>%(title)s</literal>` for the title, and `<literal>%(date)s</literal>` for the date. Doconce comes with a few templates. The usage of templates is described in a `<reference name="separate document" refuri="https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html">separate document</reference><target ids="separate-document" names="separate\ document" refuri="https://doconce.googlecode.com/hg/doc/design/wrapper_tech.html"/>`. That document describes how you your Doconce-generated

"	tutorial.xml	"
	<p>HTML file can have any specified layout.</p> <p>If the Pygments package (including the <code>pygmentize</code> program) is installed, code blocks are typeset with aid of this package. The command-line argument <code>--no-pygments-html</code> turns off the use of Pygments and makes code blocks appear with plain (<code>pre</code>) HTML tags. The option <code>--pygments-html-linenos</code> turns on line numbers in Pygments-formatted code blocks. A specific Pygments style is set by <code>--pygments-html-style=style</code>, where <code>style</code> can be <code>default</code>, <code>emacs</code>, <code>perldoc</code>, and other valid names for Pygments styles.</p> <p>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": <code>%(title)s</code> for a title, <code>%(date)s</code> for a date, and <code>%(main)s</code> 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 <code>DATE:</code> 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 <code>%(title)s</code> and <code>%(date)s</code> at appropriate places and replace the main body of text by <code>%(main)s</code>. Here is an example:</p> <pre><literal_block xml:space="preserve">Terminal> doconce format html mydoc --html-template=mytemplate.html</literal_block></section><section ids="blogs" names="blogs"><title>Blogs</title><paragraph>Doconce can be used for writing blogs provided the blog site accepts raw HTML code. Google's Blogger service (<literal>blogger.com</literal> or <literal>blogname.blogspot.com</literal>) is particularly well suited since it also allows extensive LaTeX mathematics via MathJax.</paragraph><enumerated_list enumtype="arabic" prefix="" suffix="."><list_item><paragraph>Write the blog text as a Doconce document without any title, author, and date.</paragraph></list_item><list_item><paragraph>Generate HTML as described above.</paragraph></list_item><list_item><paragraph>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.</paragraph></list_item></enumerated_list><paragraph>See a <reference name="simple blog example" refuri="http://doconce.blogspot.no">simple blog example</reference><target ids="simple-blog-example" names="simple\ blog\ example" refuri="http://doconce.blogspot.no"/> and a <reference name="scientific report" refuri="http://doconce-report-demo.blogspot.no">scientific report</reference><target dupnames="scientific\ report" ids="scientific-report" refuri="http://doconce-report-demo.blogspot.no"/> for demonstrations of blogs at <literal>blogspot.no</literal>.</paragraph><warning><paragraph>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.</paragraph></warning><system_message backrefs="id2" level="2" line="331" source="tutorial.rst" type="WARNING"><paragraph>Duplicate explicit target name: "scientific report</pre>	

tutorial.xml

WordPress (`<literal>wordpress.com</literal>`) allows raw HTML code in blogs, but has very limited LaTeX support, basically only formulas. The `<literal>--wordpress</literal>` option to `<literal>doconce</literal>` modifies the HTML code such that all equations are typeset in a way that is acceptable to WordPress.

Look at a `<reference name="simple doconce example" refuri="http://doconce.wordpress.com">simple doconce example</reference><target ids="simple-doconce-example" names="simple\ doconce\ example" refuri="http://doconce.wordpress.com"/>` and a `<reference name="scientific report" refuri="http://doconcereportdemo.wordpress.com/">scientific report</reference><target dupnames="scientific\ report" ids="id2" refuri="http://doconcereportdemo.wordpress.com/">` to see blogging with mathematics and code on WordPress.

`</section><section ids="pandoc-and-markdown" names="pandoc\ and\ markdown"><title>Pandoc and Markdown</title><paragraph>Output in Pandoc's extended Markdown format results from:</paragraph><literal_block xml:space="preserve">Terminal> doconce format pandoc mydoc</literal_block><paragraph>The name of the output file is <literal>mydoc.mkd</literal>.`

From this format one can go to numerous other formats:

`</paragraph><literal_block xml:space="preserve">Terminal> pandoc -R -t mediawiki -o mydoc.mwk --toc mydoc.mkd</literal_block><paragraph>Pandoc supports <literal>latex</literal>, <literal>html</literal>, <literal>odt</literal> (OpenOffice), <literal>docx</literal> (Microsoft Word), <literal>rtf</literal>, <literal>texinfo</literal>, to mention some. The <literal>-R</literal> option makes Pandoc pass raw HTML or LaTeX to the output format instead of ignoring it, while the <literal>--toc</literal> option generates a table of contents. See the <reference name="Pandoc documentation" refuri="http://johnmacfarlane.net/pandoc/README.html">Pandoc documentation</reference><target ids="pandoc-documentation" names="pandoc\ documentation" refuri="http://johnmacfarlane.net/pandoc/README.html"/> for the many features of the <literal>pandoc</literal> program. The HTML output from <literal>pandoc</literal> needs adjustments to provide full support for MathJax LaTeX mathematics, and for this purpose one should use <literal>doconce md2html</literal>:`

`</paragraph><literal_block xml:space="preserve">Terminal> doconce format pandoc mydoc</literal_block><paragraph>The result <literal>mydoc.html</literal> can be viewed in a browser.</paragraph><paragraph>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): <literal>doconce format pandoc</literal> and then translating using <literal>doconce md2latex</literal> (which runs <literal>pandoc</literal>), or <literal>doconce format latex</literal>, and then going from LaTeX to the desired format using <literal>pandoc</literal>.`

Here is an example on the latter strategy:

`</paragraph><literal_block xml:space="preserve">Terminal> doconce format latex mydoc</literal_block><literal_block xml:space="preserve">Terminal> doconce ptex2tex mydoc</literal_block><literal_block xml:space="preserve">Terminal> doconce replace '\Verb!' '\verb!' mydoc.tex</literal_block><literal_block xml:space="preserve">Terminal> pandoc -f latex -t docx -o mydoc.docx mydoc.tex</literal_block><paragraph>When we go through <literal>pandoc</literal>, only single equations, <literal>align</literal>, or <literal>align*</literal> environments are well understood for output to HTML.</paragraph><paragraph>Note that Doconce applies the <literal>Verb</literal> macro from the <literal>fancyvrb</literal> package`

tutorial.xml

while `<literal>pandoc</literal>` only supports the standard `<literal>verb</literal>` construction for inline verbatim text. Moreover, quite some additional `<literal>doconce` `replace</literal>` and `<literal>doconce subst</literal>` edits might be needed on the `<literal>.mkd</literal>` or `<literal>.tex</literal>` 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 my MathJax:

```
<literal_block xml:space="preserve">Terminal> doconce format pandoc mydoc
Terminal> pandoc -t html -o mydoc.html -s --mathjax mydoc.mkd</literal_block>
```

The `<literal>-s</literal>` option adds a proper header and footer to the `<literal>mydoc.html</literal>` file.

This recipe is a quick way of making HTML notes with (some) mathematics.

LaTeX

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

- Note: putting code blocks inside a list is not successful in many formats - the text may be messed up. A better choice is a paragraph environment, as used here.

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

```
<literal_block xml:space="preserve">Terminal> doconce format latex mydoc</literal_block>
```

LaTeX-specific commands (‘newcommands’ etc.) in math formulas and similar can be placed in files `<literal>newcommands.tex</literal>`, `<literal>newcommands_keep.tex</literal>`, or `<literal>newcommands_replace.tex</literal>` (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 `<literal>--latex-printed</literal>` 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 `<literal>ptex2tex</literal>` (if you have it) to make a standard LaTeX file:

```
<literal_block xml:space="preserve">Terminal> ptex2tex mydoc</literal_block>
```

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

```
<literal_block xml:space="preserve">Terminal> doconce ptex2tex mydoc</literal_block>
```

Note that Doconce generates a `<literal>.p.tex</literal>` 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:

```
<literal_block xml:space="preserve">Terminal> ptex2tex -DHELIVETICA mydoc
Terminal> doconce ptex2tex mydoc -DHELIVETICA # alternative</literal_block>
```

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 `<literal>-DLATEX_HEADING=traditional</literal>`. A separate titlepage can be generate by `<literal>-DLATEX_HEADING=titlepage</literal>`.

Preprocessor

tutorial.xml

variables to be defined or undefined are

- `<block_quote><bullet_list bullet="*"><list_item><paragraph><literal>BOOK</literal> for the "book" documentclass rather than the standard "article" class (necessary if you apply chapter headings with <literal>=</literal>)</paragraph></list_item><list_item><paragraph><literal>PALATINO</literal> for the Palatino font</paragraph></list_item><list_item><paragraph><literal>HELVETICA</literal> for the Helvetica font</paragraph></list_item><list_item><paragraph><literal>A4PAPER</literal> for A4 paper size</paragraph></list_item><list_item><paragraph><literal>A6PAPER</literal> for A6 paper size (suitable for reading PDFs on phones)</paragraph></list_item><list_item><paragraph><literal>MOVIE15</literal> for using the movie15 LaTeX package to display movies</paragraph></list_item><list_item><paragraph><literal>PREAMBLE</literal> 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)</paragraph></list_item><list_item><paragraph><literal>MINTED</literal> for inclusion of the minted package for typesetting of code with the Pygments tool (which requires <literal>latex</literal> or <literal>pdflatex</literal> to be run with the <literal>-shell-escape</literal> option)</paragraph></list_item></bullet_list></block_quote><paragraph>If you are not satisfied with the Doconce preamble, you can provide your own preamble by adding the command-line option <literal>--latex-preamble=myfile</literal>. In case <literal>myfile</literal> contains a documentclass definition, Doconce assumes that the file contains the <emphasis>complete</emphasis> preamble you want (not that all the packages listed in the default preamble are required and must be present in <literal>myfile</literal>). Otherwise, <literal>myfile</literal> is assumed to contain <emphasis>additional</emphasis> LaTeX code to be added to the Doconce default preamble.</paragraph><paragraph>The <literal>ptex2tex</literal> tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any <literal>!bc</literal> command in the Doconce source you can insert verbatim block styles as defined in your <literal>.ptex2tex.cfg</literal> file, e.g., <literal>!bc sys</literal> for a terminal session, where <literal>sys</literal> is set to a certain environment in <literal>.ptex2tex.cfg</literal> (e.g., <literal>CodeTerminal</literal>). There are about 40 styles to choose from, and you can easily add new ones.</paragraph><paragraph>Also the <literal>doconce ptex2tex</literal> command supports preprocessor directives for processing the <literal>.p.tex</literal> file. The command allows specifications of code environments as well. Here is an example:</paragraph><literal_block xml:space="preserve">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</literal_block><paragraph>Note that <literal>@</literal> must be used to separate the begin and end LaTeX commands, unless only the environment name is given (such as <literal>minted</literal> above, which implies <literal>\begin{minted}{fortran}</literal> and <literal>\en`

tutorial.xml

```

d{minted}</literal> as
begin and end for blocks inside <literal>!bc fpro</literal> and <literal>!ec</li
teral>). Specifying
<literal>envir=ans:nt</literal> means that all other environments are typeset wi
th the
<literal>anslistings.sty</literal> package, e.g., <literal>!bc cppcod</literal>
will then result in
<literal>\begin{c++}</literal>. If no environments like <literal>sys</literal>,
<literal>fpro</literal>, or the common
<literal>envir</literal> are defined on the command line, the plain <literal>\be
gin{verbatim}</literal>
and <literal>\end{verbatim}</literal> used.</paragraph><paragraph><emphasis>Step
2b (optional).</emphasis> Edit the <literal>mydoc.tex</literal> file to your ne
eds.
For example, you may want to substitute <literal>section</literal> by <literal>s
ection*</literal> 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 <literal>doconce replace</literal> and <literal>docon
ce subst</literal>
commands. The former works with substituting text directly, while the
latter performs substitutions using regular expressions.
You will use <literal>doconce replace</literal> to edit <literal>section{</litter
al> to <literal>section*</literal>:</paragraph><literal_block xml:space="preser
ve">Terminal&gt; doconce replace 'section{' 'section*' mydoc.tex</literal_block
><paragraph>For fixing the line break of a title, you may pick a word in the
title, say &quot;Using&quot;, and insert a break after than word. With
<literal>doconce subst</literal> this is easy employing regular expressions with
a group before &quot;Using&quot; and a group after:</paragraph><literal_block xm
l:space="preserve">Terminal&gt; doconce subst 'title\{((+)Using ((+)\)}' \
'title{\&lt;\&lt;\&lt;\&lt; [1.5mm] Using \&lt;\&lt;\&lt;' mydoc.tex</literal_
block><paragraph>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 <literal>doconce subst</literal> or
<literal>doconce replace</literal> commands can be put inside the script.</parag
raph><paragraph><emphasis>Step 3.</emphasis> Compile <literal>mydoc.tex</literal
>
and create the PDF file:</paragraph><literal_block xml:space="preserve">Terminal
&gt; latex mydoc
Terminal&gt; latex mydoc
Terminal&gt; makeindex mydoc      # if index
Terminal&gt; bibitem mydoc       # if bibliography
Terminal&gt; latex mydoc
Terminal&gt; dvipdf mydoc</literal_block><paragraph>If one wishes to run <litera
l>ptex2tex</literal> and use the minted LaTeX package for
typesetting code blocks (<literal>Minted_Python</literal>, <literal>Minted_Cpp</
literal>, etc., in
<literal>ptex2tex</literal> specified through the <literal>*pro</literal> and <l
iteral>*cod</literal> variables in
<literal>.ptex2tex.cfg</literal> or <literal>$HOME/.ptex2tex.cfg</literal>), the
minted LaTeX package is
needed. This package is included by running <literal>ptex2tex</literal> with th
e
<literal>-DMINTED</literal> option:</paragraph><literal_block xml:space="preserv
e">Terminal&gt; ptex2tex -DMINTED mydoc</literal_block><paragraph>In this case,
<literal>latex</literal> must be run with the
<literal>-shell-escape</literal> option:</paragraph><literal_block xml:space="pr
eserve">Terminal&gt; latex -shell-escape mydoc

```

tutorial.xml

```

Terminal> latex -shell-escape mydoc
Terminal> makeindex mydoc      # if index
Terminal> bibitem mydoc       # if bibliography
Terminal> latex -shell-escape mydoc
Terminal> dvipdf mydoc</literal_block><paragraph>When running <literal>doconce
ptex2tex mydoc enviro=minted</literal> (or other minted
specifications with <literal>doconce ptex2tex</literal>), the minted package is
automatically
included so there is no need for the <literal>-DMINTED</literal> option.</paragr
aph></section><section ids="pdflatex" names="pdflatex"><title>PDFLaTeX</title><p
aragraph>Running <literal>pdflatex</literal> instead of <literal>latex</literal>
follows almost the same steps,
but the start is:</paragraph><literal_block xml:space="preserve">Terminal> do
conce format latex mydoc</literal_block><paragraph>Then <literal>ptex2tex</liter
al> is run as explained above, and finally:</paragraph><literal_block xml:space=
"preserve">Terminal> pdflatex -shell-escape mydoc
Terminal> makeindex mydoc      # if index
Terminal> bibitem mydoc       # if bibliography
Terminal> pdflatex -shell-escape mydoc</literal_block></section><section ids=
"plain-ascii-text" names="plain\ ascii\ text"><title>Plain ASCII Text</title><p
aragraph>We can go from Doconce &quot;back to&quot; plain untagged text suitable
for viewing
in terminal windows, inclusion in email text, or for insertion in
computer source code:</paragraph><literal_block xml:space="preserve">Terminal>
doconce format plain mydoc.do.txt # results in mydoc.txt</literal_block></sec
tion><section dupnames="restructuredtext" ids="id4"><title>reStructuredText</tit
le><paragraph>Going from Doconce to reStructuredText gives a lot of possibilitie
s to
go to other formats. First we filter the Doconce text to a
reStructuredText file <literal>mydoc.rst</literal>:</paragraph><literal_block xm
l:space="preserve">Terminal> doconce format rst mydoc.do.txt</literal_block><
paragraph>We may now produce various other formats:</paragraph><literal_block xm
l:space="preserve">Terminal> rst2html.py mydoc.rst &gt; mydoc.html # html
Terminal> rst2latex.py mydoc.rst &gt; mydoc.tex # latex
Terminal> rst2xml.py mydoc.rst &gt; mydoc.xml # XML
Terminal> rst2odt.py mydoc.rst &gt; mydoc.odt # OpenOffice</literal_block>
<paragraph>The OpenOffice file <literal>mydoc.odt</literal> can be loaded into O
penOffice and
saved in, among other things, the RTF format or the Microsoft Word format.
However, it is more convenient to use the program <literal>unovonv</literal>
to convert between the many formats OpenOffice supports <emphasis>on the command
line</emphasis>.
Run:</paragraph><literal_block xml:space="preserve">Terminal> unoconv --show<
/literal_block><paragraph>to see all the formats that are supported.
For example, the following commands take
<literal>mydoc.odt</literal> to Microsoft Office Open XML format,
classic MS Word format, and PDF:</paragraph><literal_block xml:space="preserve">
Terminal> unoconv -f ooxml mydoc.odt
Terminal> unoconv -f doc mydoc.odt
Terminal> unoconv -f pdf mydoc.odt</literal_block><paragraph><emphasis>Remark
about Mathematical Typesetting.</emphasis> At the time of this writing, there i
s no easy way to go from Doconce
and LaTeX mathematics to reST and further to OpenOffice and the
&quot;MS Word world&quot;. Mathematics is only fully supported by <literal>latex
</literal> as
output and to a wide extent also supported by the <literal>sphinx</literal> outp
ut format.
Some links for going from LaTeX to Word are listed below.</paragraph><block_quot
e><bullet_list bullet="*"><list_item><paragraph><reference name="http://ubuntufo

```

tutorial.xml

```

rums.org/showthread.php?t=1033441" refuri="http://ubuntuforums.org/showthread.php?t=1033441">http://ubuntuforums.org/showthread.php?t=1033441</reference><target
ids="http-ubuntuforums-org-showthread-php-t-1033441" names="http://ubuntuforums
.org/showthread.php?t=1033441" refuri="http://ubuntuforums.org/showthread.php?t=
1033441"/></paragraph></list_item><list_item><paragraph><reference name="http://
tug.org/utilities/texconv/textopc.html" refuri="http://tug.org/utilities/texconv
/textopc.html">http://tug.org/utilities/texconv/textopc.html</reference><target
ids="http-tug-org-utilities-texconv-textopc-html" names="http://tug.org/utilitie
s/texconv/textopc.html" refuri="http://tug.org/utilities/texconv/textopc.html"/>
</paragraph></list_item><list_item><paragraph><reference name="http://nileshbans
al.blogspot.com/2007/12/latex-to-openofficeword.html" refuri="http://nileshbansa
l.blogspot.com/2007/12/latex-to-openofficeword.html">http://nileshbansal.blogspo
t.com/2007/12/latex-to-openofficeword.html</reference><target ids="http-nileshba
nsal-blogger-com-2007-12-latex-to-openofficeword-html" names="http://nileshbans
al.blogspot.com/2007/12/latex-to-openofficeword.html" refuri="http://nileshbansa
l.blogspot.com/2007/12/latex-to-openofficeword.html"/></paragraph></list_item></
bullet_list></block_quote></section><section dupnames="sphinx" ids="id5"><title>
Sphinx</title><paragraph>Sphinx documents demand quite some steps in their creat
ion. We have automated
most of the steps through the <literal>doconce sphinx_dir</literal> command:</pa
ragraph><literal_block xml:space="preserve">Terminal> doconce sphinx_dir auth
or=&quot;authors' names&quot; \
    title=&quot;some title&quot; version=1.0 dirname=sphinxdir \
    theme=mytheme file1 file2 file3 ...</literal_block><paragraph>The keyw
ords <literal>author</literal>, <literal>title</literal>, and <literal>version</
literal> are used in the headings
of the Sphinx document. By default, <literal>version</literal> is 1.0 and the sc
ript
will try to deduce authors and title from the doconce files <literal>file1</lite
ral>,
<literal>file2</literal>, etc. that together represent the whole document. Note
that
none of the individual Doconce files <literal>file1</literal>, <literal>file2</l
iteral>, etc. should
include the rest as their union makes up the whole document.
The default value of <literal>dirname</literal> is <literal>sphinx-rootdir</lite
ral>. The <literal>theme</literal>
keyword is used to set the theme for design of HTML output from
Sphinx (the default theme is <literal>'default'</literal>).</paragraph><paragrap
h>With a single-file document in <literal>mydoc.do.txt</literal> one often just
runs:</paragraph><literal_block xml:space="preserve">Terminal> doconce sphinx
_dir mydoc</literal_block><paragraph>and then an appropriate Sphinx directory <l
iteral>sphinx-rootdir</literal> is made with
relevant files.</paragraph><paragraph>The <literal>doconce sphinx_dir</literal>
command generates a script
<literal>automake_sphinx.py</literal> for compiling the Sphinx document into an
HTML
document. One can either run <literal>automake_sphinx.py</literal> or perform t
he
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 <literal>automake_sphinx.py</
literal>
file.</paragraph><paragraph><emphasis>Links.</emphasis> The <literal>automake_sp
hinx.py</literal> script copies directories named <literal>fig*</literal>
over to the Sphinx directory so that figures are accessible
in the Sphinx compilation. It also examines <literal>MOVIE:</literal> and <lite
ral>FIGURE:</literal>

```

tutorial.xml

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 `<literal>http:</literal>` or `<literal>file:</literal>` URLs) in a directory named `<literal>_static</literal>`. The `<literal>automake_sphinx.py</literal>` copies `<literal>_static*</literal>` to the Sphinx directory, which guarantees that the links to the local files will work in the Sphinx document.

There is a utility `<literal>doconce sphinxfix_localURLs</literal>` for checking links to local files and moving the files to `<literal>_static</literal>` and changing the links accordingly. For example, a link to `<literal>dir1/dir2/myfile.txt</literal>` is changed to `<literal>_static/myfile.txt</literal>` and `<literal>myfile.txt</literal>` is copied to `<literal>_static</literal>`.

However, I recommend instead that you manually copy files to `<literal>_static</literal>` 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 `<literal>agni</literal>`, `<literal>basicstrap</literal>`, `<literal>bootstrap</literal>`, `<literal>cloud</literal>`, `<literal>fenics</literal>`, `<literal>fenics_minimal</literal>`, `<literal>flask</literal>`, `<literal>haiku</literal>`, `<literal>impressjs</literal>`, `<literal>jal</literal>`, `<literal>pylons</literal>`, `<literal>redcloud</literal>`, `<literal>scipy_lectures</literal>`, `<literal>slim-agogo</literal>`, and `<literal>vlinux-theme</literal>`.

All the themes are packed out in the Sphinx directory, and the `<literal>doconce sphinx_dir</literal>` insert lots of extra code in the `<literal>conf.py</literal>` 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 `<literal>conf.py</literal>` file is a good starting point for fine-tuning your favorite team, and your own `<literal>conf.py</literal>` file can later be supplied and used when running `<literal>doconce sphinx_dir</literal>`: simply add the command-line option `<literal>conf.py=conf.py</literal>`.

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

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

The resulting directories with HTML documents are `<literal>_build/html_fenics</literal>` and `<literal>_build/html_pyramid</literal>`, respectively. Without arguments,

tutorial.xml

```

<literal>make-themes.sh</literal> makes all available themes (!). With <literal>
make-themes.sh</literal>
it is easy to check out various themes to find the one that is most
attractive for your document.</paragraph><paragraph>You may supply your own them
e and avoid copying all the themes
that come with Doconce into the Sphinx directory. Just specify
<literal>theme_dir=path</literal> on the command line, where <literal>path</lite
ral> is the relative
path to the directory containing the Sphinx theme. You must also
specify a configure file by <literal>conf.py=path</literal>, where <literal>path
</literal> is the
relative path to your <literal>conf.py</literal> file.</paragraph><paragraph><em
phasis>Example.</emphasis> Say you like the <literal>scipy_lectures</literal> th
eme, but you want
a table of contents to appear <emphasis>to the right</emphasis>, much in the sam
e style
as in the <literal>default</literal> theme (where the table of contents is to th
e left).
You can then run <literal>doconce sphinx_dir</literal>, invoke a text editor wit
h the
<literal>conf.py</literal> file, find the line <literal>html_theme == 'scipy_lec
tures'</literal>,
edit the following <literal>nosidebar</literal> to <literal>>false</literal> and
<literal>rightsidebar</literal> to <literal>>true</literal>.
Alternatively, you may write a little script using <literal>doconce replace</lit
eral>
to replace a portion of text in <literal>conf.py</literal> by a new one:</paragr
aph><literal_block xml:space="preserve">doconce replace &quot;elif html_theme ==
'scipy_lectures':
    html_theme_options = {
        'nosidebar': 'true',
        'rightsidebar': 'false',
        'sidebarbgcolor': '#f2f2f2',
        'sidebartextcolor': '#20435c',
        'sidebarlinkcolor': '#20435c',
        'footerbgcolor': '#000000',
        'relbarbgcolor': '#000000',
    }&quot; &quot;elif html_theme == 'scipy_lectures':
    html_theme_options = {
        'nosidebar': 'false',
        'rightsidebar': 'true',
        'sidebarbgcolor': '#f2f2f2',
        'sidebartextcolor': '#20435c',
        'sidebarlinkcolor': '#20435c',
        'footerbgcolor': '#000000',
        'relbarbgcolor': '#000000',
    }&quot; conf.py</literal_block><paragraph>Obviously, we could also have chan
ged colors in the edit above.
The final alternative is to save the edited <literal>conf.py</literal> file some
where
and reuse it the next time <literal>doconce sphinx_dir</literal> is run:</paragr
aph><literal_block xml:space="preserve">doconce sphinx_dir theme=scipy_lectures
\
    conf.py=../some/path/conf.py mydoc</literal_block><section id
s="the-manual-sphinx-procedure" names="the\ manual\ sphinx\ procedure"><title>Th
e manual Sphinx procedure</title><paragraph>If it is not desirable to use the au
togenerated scripts explained
above, here is the complete manual procedure of generating a
Sphinx document from a file <literal>mydoc.do.txt</literal>.</paragraph><paragra

```

tutorial.xml

```

ph><emphasis>Step 1.</emphasis> Translate Doconce into the Sphinx format:</parag
raph><literal_block xml:space="preserve">Terminal> doconce format sphinx mydo
c</literal_block><paragraph><emphasis>Step 2.</emphasis> Create a Sphinx root di
rectory
either manually or by using the interactive <literal>sphinx-quickstart</literal>
program. Here is a scripted version of the steps with the latter:</paragraph><li
teral_block xml:space="preserve">mkdir sphinx-rootdir
sphinx-quickstart &lt;&lt;&lt;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</literal_block><paragraph>The autogenerated <literal>conf.py</literal> file
may need some edits if you want to specific layout (Sphinx themes)
of HTML pages. The <literal>doconce sphinx_dir</literal> generator makes an exte
nded <literal>conv.py</literal>
file where, among other things, several useful Sphinx extensions
are included.</paragraph><paragraph><emphasis>Step 3.</emphasis> Copy the <liter
al>mydoc.rst</literal> file to the Sphinx root directory:</paragraph><literal_bl
ock xml:space="preserve">Terminal> cp mydoc.rst sphinx-rootdir</literal_block>
<paragraph>If you have figures in your document, the relative paths to those wi
ll
be invalid when you work with <literal>mydoc.rst</literal> in the <literal>sphin
x-rootdir</literal>
directory. Either edit <literal>mydoc.rst</literal> so that figure file paths ar
e correct,
or simply copy your figure directories to <literal>sphinx-rootdir</literal>.
Links to local files in <literal>mydoc.rst</literal> must be modified to links t
o
files in the <literal>_static</literal> directory, see comment above.</paragraph>
<paragraph><emphasis>Step 4.</emphasis> Edit the generated <literal>index.rst</
literal> file so that <literal>mydoc.rst</literal>
is included, i.e., add <literal>mydoc</literal> to the <literal>toctree</literal>
> section so that it becomes:</paragraph><literal_block xml:space="preserve">..
toctree::
    :maxdepth: 2

    mydoc</literal_block><paragraph>(The spaces before <literal>mydoc</literal> a
re important!)</paragraph><paragraph><emphasis>Step 5.</emphasis> Generate, for
instance, an HTML version of the Sphinx source:</paragraph><literal_block xml:sp
ace="preserve">make clean    # remove old versions
make html</literal_block><paragraph>Sphinx can generate a range of different for

```


tutorial.xml

mats:
standalone HTML, HTML in separate directories with `<literal>index.html</literal>` 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 `<literal>!bc</literal>`: `<literal>cod</literal>` gives Python (`<literal>code-block:: python</literal>` in Sphinx syntax) and `<literal>cppcod</literal>` 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:

`<reference name="Googlecode wiki" refuri="http://code.google.com/p/support/wiki/WikiSyntax">Googlecode wiki</reference>`
`<target ids="googlecode-wiki" names="googlecode\ wiki" refuri="http://code.google.com/p/support/wiki/WikiSyntax"/>`,
`<reference name="MediaWiki" refuri="http://www.mediawiki.org/wiki/Help:Formatting">MediaWiki</reference>`
`<target ids="mediawiki" names="mediawiki" refuri="http://www.mediawiki.org/wiki/Help:Formatting"/>`, and
`<reference name="Creole Wiki" refuri="http://www.wikicreole.org/wiki/Creole1.0">Creole Wiki</reference>`
`<target ids="creole-wiki" names="creole\ wiki" refuri="http://www.wikicreole.org/wiki/Creole1.0"/>`.

These formats are called `<literal>gwiki</literal>`, `<literal>mwiki</literal>`, and `<literal>cwiki</literal>`, 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 `<reference name="sandbox of wikibooks.org" refuri="http://en.wikibooks.org/wiki/Sandbox">sandbox of wikibooks.org</reference>`
`<target ids="sandbox-of-wikibooks-org" names="sandbox\ of\ wikibooks.org" refuri="http://en.wikibooks.org/wiki/Sandbox"/>`. The format works well with Wikipedia, Wikibooks, and

`<reference name="ShoutWiki" refuri="http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page">ShoutWiki</reference>`
`<target ids="shoutwiki" names="shoutwiki" refuri="http://doconcedemo.shoutwiki.com/wiki/Doconce_demo_page"/>`, but not always well elsewhere (see `<reference name="this example" refuri="http://doconcedemo.jumpwiki.com/wiki/First_demo">this example</reference>`
`<target ids="this-example" names="this\ example" refuri="http://doconcedemo.jumpwiki.com/wiki/First_demo"/>`).

Large MediaWiki documents can be made with the `<reference name="Book creator" refuri="http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator">Book creator</reference>`
`<target ids="book-creator" names="book\ creator" refuri="http://en.wikipedia.org/w/index.php?title=Special:Book&bookcmd=book_creator"/>`.

From the MediaWiki format one can go to other formats with aid of `<reference name="mwlib" refuri="http://pediapress.com/code/">mwlib</reference>`
`<target ids="mwlib" names="mwlib" refuri="http://pediapress.com/code/">`. This means that one can easily use Doconce to write `<reference name="Wikibooks" refuri="http://en.wikibooks.org">Wikibooks</reference>`
`<target ids="wikibooks" names="wikibooks" refuri="http://en.wikibooks.org"/>` and publish these in PDF and MediaWiki format, while at the same time, the book can also be published as a

” **tutorial.xml** ”

standard LaTeX book, a Sphinx web document, or a collection of HTML files.

The Googlecode wiki document, `<literal>mydoc.gwiki</literal>`, 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 `<literal>.gwiki</literal>` 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 `<literal>.rst</literal>` file is going to be filtered to LaTeX or HTML, it cannot know if `<literal>.eps</literal>` or `<literal>.png</literal>` 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 `<literal>make.sh</literal>` files in `<literal>docs/manual</literal>` and `<literal>docs/tutorial</literal>` 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 `<literal>make.sh</literal>` in the `<literal>tutorial</literal>` directory of the Doconce source code contains a demo of how to produce a variety of formats. The source of this tutorial, `<literal>tutorial.do.txt</literal>` is the starting point. Running `<literal>make.sh</literal>` and studying the various generated files and comparing them with the original `<literal>tutorial.do.txt</literal>` file, gives a quick introduction to how Doconce is used in a real case.

[Here](https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html) [Here](https://doconce.googlecode.com/hg/doc/demos/tutorial/index.html) is a sample of how this tutorial looks in different formats.

There is another demo in the `<literal>docs/manual</literal>` directory which translates the more comprehensive documentation, `<literal>manual.do.txt</literal>`, to various formats. The `<literal>make.sh</literal>` 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 (`<literal>hg</literal>`) source follows the standard procedure:

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

tutorial.xml

```

cd ../literal_block><paragraph>Since Doconce is frequently updated, it is recom
mended to use the
above procedure and whenever a problem occurs, make sure to
update to the most recent version:</paragraph><literal_block xml:space="preserve
">cd doconce
hg pull
hg update
sudo python setup.py install</literal_block><paragraph>Debian GNU/Linux users ca
n also run:</paragraph><literal_block xml:space="preserve">sudo apt-get install
doconce</literal_block><paragraph>This installs the latest release and not the m
ost updated and bugfixed
version.
On Ubuntu one needs to run:</paragraph><literal_block xml:space="preserve">sudo
add-apt-repository ppa:scitools/ppa
sudo apt-get update
sudo apt-get install doconce</literal_block></section><section ids="dependencies"
" names="dependencies"><title>Dependencies</title><section ids="preprocessors" n
ames="preprocessors"><title>Preprocessors</title><paragraph>If you make use of t
he <reference name="Preprocess" refuri="http://code.google.com/p/preprocess">Pre
process</reference><target ids="preprocess" names="preprocess" refuri="http://co
de.google.com/p/preprocess"/>
preprocessor, this program must be installed:</paragraph><literal_block xml:spac
e="preserve">svn checkout http://preprocess.googlecode.com/svn/trunk/ preprocess
cd preprocess
cd doconce
sudo python setup.py install
cd ../literal_block><paragraph>A much more advanced alternative to Preprocess i
s
<reference name="Mako" refuri="http://www.makotemplates.org">Mako</reference><ta
rget ids="mako" names="mako" refuri="http://www.makotemplates.org"/>. Its instal
lation is most
conveniently done by <literal>pip</literal>:</paragraph><literal_block xml:space
="preserve">pip install Mako</literal_block><paragraph>This command requires <li
teral>pip</literal> to be installed. On Debian Linux systems,
such as Ubuntu, the installation is simply done by:</paragraph><literal_block xm
l:space="preserve">sudo apt-get install python-pip</literal_block><paragraph>Alt
ernatively, one can install from the <literal>pip</literal> <reference name="sou
rce code" refuri="http://pypi.python.org/pypi/pip">source code</reference><target
ids="source-code" names="source\ code" refuri="http://pypi.python.org/pypi/pip
"/>.</paragraph><paragraph>Mako can also be installed directly from
<reference name="source" refuri="http://www.makotemplates.org/download.html">sou
rce</reference><target ids="source" names="source" refuri="http://www.makotempla
tes.org/download.html"/>: download the
tarball, pack it out, go to the directory and run
the usual <literal>sudo python setup.py install</literal>.</paragraph></section>
<section ids="image-file-handling" names="image\ file\ handling"><title>Image fi
le handling</title><paragraph>Different output formats require different formats
of image files.
For example, PostScript or Encapsulated PostScript is required for <literal>late
x</literal>
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 <reference name="ImageMagick suite
" refuri="http://www.imagemagick.org/script/index.php">ImageMagick suite</refere
nce><target ids="imagemagick-suite" names="imagemagick\ suite" refuri="http://ww
w.imagemagick.org/script/index.php"/> can be installed on all major platforms.
On Debian Linux (including Ubuntu) systems one can simply write:</paragraph><lit
eral_block xml:space="preserve">sudo apt-get install imagemagick</literal_block>
<paragraph>The convenience program <literal>doconce combine_images</literal>, fo

```

tutorial.xml

r combining several images into one, will use `<literal>montage</literal>` and `<literal>convert</literal>` from ImageMagick and the `<literal>pdftk</literal>`, `<literal>pdfnup</literal>`, and `<literal>pdfcrop</literal>` programs from the `<literal>texlive-extra-utils</literal>` Debian package. The latter gets installed by:

```
<literal_block xml:space="preserve">sudo apt-get install texlive-extra-utils</literal_block></section><section ids="spellcheck" names="spellcheck"><title>Spellcheck</title><paragraph>The utility <literal>doconce spellcheck</literal> applies the <literal>ispell</literal> program for spellcheck. On Debian (including Ubuntu) it is installed by:</paragraph><literal_block xml:space="preserve">sudo apt-get install ispell</literal_block></section><section ids="ptex2tex-for-latex-output" names="ptex2tex\ for\ latex\ output"><title>Ptex2tex for LaTeX Output</title><paragraph>To make LaTeX documents with v
```

ery flexible choice of typesetting of verbatim code blocks you need `<reference name="ptex2tex" refuri="http://code.google.com/p/ptex2tex">ptex2tex</reference><target ids="ptex2tex" names="ptex2tex" refuri="http://code.google.com/p/ptex2tex"/>`, which is installed by:

```
<literal_block xml:space="preserve">svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex</literal_block></section><section ids="ptex2tex-setup" names="ptex2tex\ setup"><title>Ptex2tex setup</title><paragraph>To make LaTeX documents with v
```

```
cd ptex2tex</literal_block></section><section ids="ptex2tex-stylefiles" names="ptex2tex\ stylefiles"><title>Ptex2tex stylefiles</title><paragraph>It may happen that you need additional style files, you can run a script, <literal>cp2texmf.sh</literal>:</paragraph><literal_block xml:space="preserve">cd latex</literal_block></section><section ids="ptex2tex-stylefiles-2" names="ptex2tex\ stylefiles"><title>Ptex2tex stylefiles</title><paragraph>This script copies some special stylefiles that at that <literal>ptex2tex</literal> potentially makes use of. Some more standard stylefiles are also needed. These are installed by:</paragraph><literal_block xml:space="preserve">sudo apt-get install texlive-latex-recommended texlive-latex-extra</literal_block></section><section ids="ptex2tex-stylefiles-3" names="ptex2tex\ stylefiles"><title>Ptex2tex stylefiles</title><paragraph>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 <literal>~/texmf/tex/latex/misc</literal> directory).</paragraph><paragraph>Note that the <literal>doconce ptex2tex</literal> command, which needs no installation beyond Doconce itself, can be used as a simpler alternative to the <literal>ptex2tex</literal> program.</paragraph><paragraph>The <emphasis>minted</emphasis> LaTeX style is offered by <literal>ptex2tex</literal> and <literal>doconce ptext2tex</literal> and popular among many users. This style requires the package <reference name="Pygments" refuri="http://pygments.org">Pygments</reference><target ids="pygments" names="pygments" refuri="http://pygments.org"/> to be installed. On Debian Linux:</paragraph><literal_block xml:space="preserve">sudo apt-get install python-pygments</literal_block></section><section ids="ptex2tex-stylefiles-4" names="ptex2tex\ stylefiles"><title>Ptex2tex stylefiles</title><paragraph>Alternatively, the package can be installed manually:</paragraph><literal_block xml:space="preserve">hg clone ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments</literal_block></section><section ids="ptex2tex-stylefiles-5" names="ptex2tex\ stylefiles"><title>Ptex2tex stylefiles</title><paragraph>If you use the minted style together with <literal>ptex2tex</literal>, you have to enable it by the <literal>-DMINTED</literal> command-line argument to <literal>ptex2tex</literal>.
```

This is not necessary if you run the alternative `<literal>doconce ptex2tex</literal>` program.

All use of the minted style requires the `<literal>-shell-escape</literal>` command-line

tutorial.xml

```

argument when running LaTeX, i.e., <literal>latex -shell-escape</literal> or <li
teral>pdflatex
-shell-escape</literal>.</paragraph><comment xml:space="preserve">Say something
about anslistings.sty</comment></section><section ids="restructuredtext-rest-out
put" names="restructuredtext\ (rest)\ output"><title>reStructuredText (reST) Out
put</title><paragraph>The <literal>rst</literal> output from Doconce allows furt
her transformation to LaTeX,
HTML, XML, OpenOffice, and so on, through the <reference name="docutils" refuri=
"http://docutils.sourceforge.net">docutils</reference><target ids="docutils" nam
es="docutils" refuri="http://docutils.sourceforge.net"/> package. The installat
ion of the
most recent version can be done by:</paragraph><literal_block xml:space="preserv
e">svn checkout http://docutils.svn.sourceforge.net/svnroot/docutils/trunk/docut
ils
cd docutils
sudo python setup.py install
cd ..</literal_block><paragraph>To use the OpenOffice suite you will typically o
n Debian systems install:</paragraph><literal_block xml:space="preserve">sudo ap
t-get install unovonv libreoffice libreoffice-dmaths</literal_block><paragraph>T
here is a possibility to create PDF files from reST documents
using ReportLab instead of LaTeX. The enabling software is
<reference name="rst2pdf" refuri="http://code.google.com/p/rst2pdf">rst2pdf</ref
erence><target ids="rst2pdf" names="rst2pdf" refuri="http://code.google.com/p/rs
t2pdf"/>. Either download the tarball
or clone the svn repository, go to the <literal>rst2pdf</literal> directory and
run the usual <literal>sudo python setup.py install</literal>.</paragraph><parag
raph>Output to <literal>sphinx</literal> requires of course the
<reference name="Sphinx software" refuri="http://sphinx.pocoo.org">Sphinx softwa
re</reference><target ids="sphinx-software" names="sphinx\ software" refuri="htt
p://sphinx.pocoo.org"/>,
installed by:</paragraph><literal_block xml:space="preserve">hg clone https://bi
tbucket.org/birkenfeld/sphinx
cd sphinx
sudo python setup.py install
cd ..</literal_block></section><section ids="markdown-and-pandoc-output" names="
markdown\ and\ pandoc\ output"><title>Markdown and Pandoc Output</title><paragra
ph>The Doconce format <literal>pandoc</literal> outputs the document in the Pand
oc
extended Markdown format, which via the <literal>pandoc</literal> program can be
translated to a range of other formats. Installation of <reference name="Pandoc"
refuri="http://johnmacfarlane.net/pandoc/">Pandoc</reference><target ids="pando
c" names="pandoc" refuri="http://johnmacfarlane.net/pandoc/">, written in Haske
ll, is most
easily done by:</paragraph><literal_block xml:space="preserve">sudo apt-get inst
all pandoc</literal_block><paragraph>on Debian (Ubuntu) systems.</paragraph></se
ction><section ids="epydoc-output" names="epydoc\ output"><title>Epydoc Output</
title><paragraph>When the output format is <literal>epydoc</literal> one needs t
hat program too, installed
by:</paragraph><literal_block xml:space="preserve">svn co https://epydoc.svn.sou
rceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
sudo make install
cd ..</literal_block><paragraph><emphasis>Remark.</emphasis> Several of the pack
ages above installed from source code
are also available in Debian-based system through the
<literal>apt-get install</literal> command. However, we recommend installation d
irectly
from the version control system repository as there might be important
updates and bug fixes. For <literal>svn</literal> directories, go to the directo

```

”

tutorial.xml

”

```
ry,  
run <literal>svn update</literal>, and then <literal>sudo python setup.py instal  
l</literal>. For  
Mercurial (<literal>hg</literal>) directories, go to the directory, run  
<literal>hg pull; hg update</literal>, and then <literal>sudo python setup.py in  
stall</literal>.</paragraph></section></section></section></document>
```