tutorial.do.txt

```
TITLE: Doconce: Document Once, Include Anywhere
AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of O
slo
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/LaTeXRefCa
rd.v2.0.pdf", "HTML":"http://www.htmlcodetutorial.com/", "reStructuredText":"htt
p://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.


======= The Doconce Concept  =======

# #include "_what_is.do.txt"

======= 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 formating. Here are
som examples.

  * Bullet lists arise from lines starting with an asterisk.

  * *Emphasized words* are surrounded by asterisks.

  * _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 recognied 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 special support for advanced exercises features.

  * With a simple 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 the Mako preprocessor one can even embed Python
    code and use this to steer generation of Doconce text.

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 an 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.do.txt

```
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 supperted, 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
!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 supperted, 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    |
  |--------------------------------|

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

Inline mathematics, such as $\nu = \sin(x)$|$v = sin(x)$,
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 $\nu = \sin(x)$|$v = sin(x)$ 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.

Blocks of mathematics are better typeset with raw LaTeX, inside
‘!bt‘ and ‘!et‘ (begin tex / end tex) instructions.
The result looks like this:
!bt
\begin{eqnarray}
{\partial u\over\partial t} &=& \nabla^2 u + f, label{myeq1}\\
{\partial v\over\partial t} &=& \nabla\cdot(q(u)\nabla v) + g
\end{eqnarray}
!et
Of course, such blocks only looks nice in LaTeX. The raw
LaTeX syntax appears in all other 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. Such blocks look like
!bc cod
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.

It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., ‘!bc xxx‘
where ‘xxx‘ is an identifier like ‘pycod‘ for code snippet in Python,
‘sys‘ for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in ‘ptex2tex‘ and defined in a
configuration file ‘.ptext2tex.cfg‘, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):
!bc
# sphinx code-blocks: pycod=python cod=fortran cppcod=c++ sys=console
!ec
By default, ‘pro‘ and ‘cod‘ are ‘python‘, ‘sys‘ is ‘console‘,
while ‘xpro‘ and ‘xcod‘ are computer language specific for ‘x‘
in ‘f‘ (Fortran), ‘c‘ (C), ‘cpp‘ (C++), ‘pl‘ (Perl), ‘m‘ (Matlab),
```

```
‘sh‘ (Unix shells), ‘cy‘ (Cython), and ‘py‘ (Python).

# (Any sphinx code-block comment, whether inside verbatim code
# blocks or outside, yields a mapping between bc arguments
# and computer languages. In case of muliple definitions, the
# first one is used.)

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!). A complete file is typeset
with ‘!bc pro‘, while a part of a file is copied into a ‘!bc cod‘
environment. What ‘pro‘ and ‘cod‘ mean is then defined through
a ‘.ptex2tex.cfg‘ file for LaTeX and a ‘sphinx code-blocks‘
comment for Sphinx.

Another document can be included by writing ‘#include "mynote.do.txt"‘
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.


===== 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).

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).
```

tutorial.do.txt

```
# 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]

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

Jul 25, 2012

- 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.

1 The Doconce Concept

Doconce is two things:

1. Doconce is a very simple and minimally tagged markup language that looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reST you can (via `rst2*` programs) go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter (via `unoconv`) to RTF, numerous MS Word formats (including MS Office Open XML), DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU Texinfo, MediaWiki, RTF, Groff, and other formats.

2. Doconce is a working strategy for never duplicating information. Text is written in a single place and then transformed to a number of different destinations of diverse type (software source code, manuals, tutorials, books, wikis, memos, emails, etc.). The Doconce markup language support this working strategy. The slogan is: "Write once, include anywhere".

Here are some Doconce features:

- Doconce markup does include tags, so the format is more tagged than Markdown and Pandoc, but less than reST, and very much less than LaTeX and HTML.
- Doconce can be converted to plain *untagged* text, often desirable for computer programs and email.
- Doconce has good support for copying in parts of computer code directly from the source code files via regular expressions for the start and end lines.
- Doconce has full support for LaTeX math and integrates well with big LaTeX projects (books).
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
- Contrary to the similar (and superior) Pandoc translator, Doconce supports Sphinx, Google wiki, Creole wiki (for bitbucket.org), lots of computer code environments in LaTeX, and a special exercise syntax. Doconce also also runs preprocessors (including Mako) such that the author can mix ordinary text with programming construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

- Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
- Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at web sites, and as LaTeX integrated in, e.g., a thesis.
- Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover, Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting and handling text through regular expressions. The possibility for tweaking the layout is obviously limited since the text can go to all sorts of sophisticated markup languages. Moreover, because of limitations of regular expressions, some formatting of Doconce syntax may face problems when transformed to HTML, LaTeX, Sphinx, and similar formats.

2 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 formating. Here are som examples.

- Bullet lists arise from lines starting with an asterisk.
- *Emphasized words* are surrounded by asterisks.
- **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 recognied 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 special support for advanced exercises features.
- With a simple 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 the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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 an 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 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 supperted, 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 Section 2.1.

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 3 for an example).

Tables are also supperted, 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

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

The pipe symbol acts as a delimiter between LaTeX code and the plain text version of the formula.

Blocks of mathematics are better typeset with raw LaTeX, inside bt! and et! (begin tex / end tex) instructions. 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 looks nice in LaTeX. The raw LaTeX syntax appears in all other 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. Such blocks look like

```
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.

It is possible to add a specification of an environment for typesetting the verbatim code block, e.g., bc xxx! where `xxx` is an identifier like `pycod` for code snippet in Python, `sys` for terminal session, etc. When Doconce is filtered to LaTeX, these identifiers are used as in `ptex2tex` and defined in a configuration file `.ptext2tex.cfg`, while when filtering to Sphinx, one can have a comment line in the Doconce file for mapping the identifiers to legal language names for Sphinx (which equals the legal language names for Pygments):

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

By default, `pro` and `cod` are `python`, `sys` is `console`, while `xpro` and `xcod` are computer language specific for `x` in `f` (Fortran), `c` (C), `cpp` (C++), `pl` (Perl), `m` (Matlab), `sh` (Unix shells), `cy` (Cython), and `py` (Python).

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!). A complete file is typeset with bc pro!, while a part of a file is copied into a bc cod! environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. 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 the right writing enviroment for plain text.

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).

3 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
```

The `mako` or `preprocess` programs are always used to preprocess the file first, and options to `mako` or `preprocess` 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`. 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"`.

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.

3.1 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.

3.2 Pandoc

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 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. See the Pandoc documentation for the many features of the `pandoc` program.

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 `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> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

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

Quite some `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

```
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 makeing HTML notes with (some) mathematics.

3.3 LaTeX

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

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

Step 2. Run `ptex2tex` (if you have it) to make a standard LaTeX 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 LaTeX 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 generate by `-DLATEX_HEADING=titlepage`.

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 cod! for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}`). 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` or `fpro` are defined, 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. Here are some examples:

Terminal

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
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 encourged to make a script for generating PDF from the LaTeX file.

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 use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code (specified, e.g., in the `*pro` and `*cod` environments in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the `minted` LaTeX package is needed. This package is included by running `doconce format` 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
```

3.4 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
```

3.5 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
```

3.6 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 `unovonv` 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`

3.7 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=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

```
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. You should at least read the script prior to executing it to have some idea of what is done.

The `doconce sphinx_dir` script copies directories named `figs` or `figures` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. If figures or movies are located in other directories, `automake-sphinx.py` must be edited accordingly. Files, to which there are local links (not `http:` or `file:` URLs), must be placed in the `_static` subdirectory of the Sphinx directory. The utility `doconce sphinxfix_localURLs` is run to check for local links in the Doconce file: for each such link, say `dir1/dir2/myfile.txt` it replaces the link by `_static/myfile.txt` and copies `dir1/dir2/myfile.txt` to a local `_static` directory (in the same directory as the script is run). However, we recommend instead that the writer of the document places files in `_static` or lets a script do it automatically. The user must copy all `_static/*` files to the `_static` subdirectory of the Sphinx directory. It may be wise to always put files, to which there are local links in the Doconce document, in a `_static` or `_static-name` directory and use these local links. Then links do not need to be modified when creating a Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents. These are packed out in the Sphinx directory, the `conf.py` configuration file for Sphinx is edited accordingly, and a script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics` and `pyramid`, one writes

Terminal

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

The resulting directories with HTML documents are `_build/html_fenics` and `_build/html_pyramid`, respectively. Without arguments, `make-themes.sh` makes all available themes (!).

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 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.

3.8 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 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).

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. At the same time, the book can also be published as a standard LaTeX book or a Sphinx web document.

3.9 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.

3.10 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.

3.11 Dependencies and Installation

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
```

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.

To make LaTeX documents (without going through the reStructuredText format) you need ptex2tex, which is installed by

Terminal

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
```

As seen, `cp2texmf.sh` 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-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).

The *minted* LaTeX style is offered by `ptex2tex` and popular among users. This style requires the package Pygments:

Terminal

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

If you use the minted style, you have to enable it by running `ptex2tex -DMINTED` and then `latex -shell-escape`, see the Section 3.

For `rst` output and further transformation to LaTeX, HTML, XML, OpenOffice, and so on, one needs docutils. The installation 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 `sudo python setup.py install`.

Output to `sphinx` requires of course Sphinx, installed by

Terminal

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

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

Terminal

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

Finally, translation to `pandoc` requires the Pandoc program (written in Haskell) to be installed.

Terminal

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

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 itself is frequently updated so these commands should be run regularly.

Doconce: Document Once, Include Anywhere

Author: Hans Petter Langtangen

Date: Jul 25, 2012

- 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.

The Doconce Concept

Doconce is two things:

1. Doconce is a very simple and minimally tagged markup language that looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reST you can (via `rst2*` programs) go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter (via `unoconv`) to RTF, numerous MS Word formats (including MS Office Open XML), DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU Texinfo, MediaWiki, RTF, Groff, and other formats.
2. Doconce is a working strategy for never duplicating information. Text is written in a single place and then transformed to a number of different destinations of diverse type (software source code, manuals, tutorials, books, wikis, memos, emails, etc.). The Doconce markup language support this working strategy. The slogan is: "Write once, include anywhere".

Here are some Doconce features:

- Doconce markup does include tags, so the format is more tagged than Markdown and Pandoc, but less than reST, and very much less than LaTeX and HTML.
- Doconce can be converted to plain *untagged* text, often desirable for computer programs and email.
- Doconce has good support for copying in parts of computer code directly from the source code files via regular expressions for the start and end lines.
- Doconce has full support for LaTeX math and integrates well with big LaTeX projects (books).
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
- Contrary to the similar (and superior) Pandoc translator, Doconce supports Sphinx, Google wiki, Creole wiki (for bitbucket.org), lots of computer code environments in LaTeX, and a special exercise syntax. Doconce also also runs preprocessors (including Mako) such that the author can mix ordinary text with programming construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

- Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
- Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at web sites, and as LaTeX integrated in, e.g., a thesis.
- Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover, Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting and handling text through regular expressions. The possibility for tweaking the layout is obviously limited since the text can go to all sorts of sophisticated markup languages. Moreover, because of limitations of regular expressions, some formatting of Doconce syntax may face problems when transformed to HTML, LaTeX, Sphinx, and similar formats.

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 formating. Here are som examples.

- Bullet lists arise from lines starting with an asterisk.
- *Emphasized words* are surrounded by asterisks.
- **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 recognied 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 special support for advanced exercises features.
- With a simple 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 the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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 an 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 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 supperted, 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:

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 supperted, 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.

Blocks of mathematics are better typeset with raw LaTeX, inside `!bt` and `!et` (begin tex / end tex) instructions. The result looks like this:

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

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other 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. Such blocks look like:

```
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.

It is possible to add a specification of an environment for typesetting the verbatim code block, e.g., `!bc xxx` where `xxx` is an identifier like `pycod` for code snippet in Python, `sys` for terminal session, etc. When Doconce is filtered to LaTeX, these identifiers are used as in `ptex2tex` and defined in a configuration file `.ptext2tex.cfg`, while when filtering to Sphinx, one can have a comment line in the Doconce file for mapping the identifiers to legal language names for Sphinx (which equals the legal language names for Pygments):

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

By default, `pro` and `cod` are `python`, `sys` is `console`, while `xpro` and `xcod` are computer language specific for `x` in `f` (Fortran), `c` (C), `cpp` (C++), `pl` (Perl), `m` (Matlab), `sh` (Unix shells), `cy` (Cython), and `py` (Python).

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!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. 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 the right writing enviroment for plain text.

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
```

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

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

The variable `FORMAT` is always defined as the current format when running `preprocess`. 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"`.

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.

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.

Pandoc

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 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. See the Pandoc documentation for the many features of the `pandoc` program.

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 `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> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

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

Quite some `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 makeing 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.

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`.

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 cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}`). 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` or `fpro` are defined, 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. Here are some examples:

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
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 encourged to make a script for generating PDF from the LaTeX file.

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 use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code (specified, e.g., in the `*pro` and `*cod` environments in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the `minted` LaTeX package is needed. This package is included by running `doconce format` 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
```

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. You should at least read the script prior to executing it to have some idea of what is done.

The `doconce sphinx_dir` script copies directories named `figs` or `figures` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. If figures or movies are located in other directories, `automake-sphinx.py` must be edited accordingly. Files, to which there are local links (not `http:` or `file:`

URLs), must be placed in the `_static` subdirectory of the Sphinx directory. The utility `doconce sphinxfix_localURLs` is run to check for local links in the Doconce file: for each such link, say `dir1/dir2/myfile.txt` it replaces the link by `_static/myfile.txt` and copies `dir1/dir2/myfile.txt` to a local `_static` directory (in the same directory as the script is run). However, we recommend instead that the writer of the document places files in `_static` or lets a script do it automatically. The user must copy all `_static/*` files to the `_static` subdirectory of the Sphinx directory. It may be wise to always put files, to which there are local links in the Doconce document, in a `_static` or `_static-name` directory and use these local links. Then links do not need to be modified when creating a Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents. These are packed out in the Sphinx directory, the `conf.py` configuration file for Sphinx is edited accordingly, and a script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics` and `pyramid`, one writes:

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

The resulting directories with HTML documents are `_build/html_fenics` and `_build/html_pyramid`, respectively. Without arguments, `make-themes.sh` makes all available themes (!).

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 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).

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. At the same time, the book can also be published as a standard LaTeX book or a Sphinx web document.

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.

Dependencies and Installation

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
```

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.

To make LaTeX documents (without going through the reStructuredText format) you need ptex2tex, which is installed by:

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
```

As seen, `cp2texmf.sh` 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-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).

The *minted* LaTeX style is offered by `ptex2tex` and popular among users. This style requires the package Pygments:

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

If you use the minted style, you have to enable it by running `ptex2tex -DMINTED` and then `latex -shell-escape`, see the the section From Doconce to Other Formats.

For `rst` output and further transformation to LaTeX, HTML, XML, OpenOffice, and so on, one needs docutils. The installation can be done by:

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

system-message

WARNING/2 in `tutorial.rst`, line 403
Duplicate explicit target name: “sphinx”.

Output to `sphinx` requires of course Sphinx, installed by:

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

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

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

Finally, translation to `pandoc` requires the Pandoc program (written in Haskell) to be installed:

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

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 itself is frequently updated so these commands should be run regularly.

Doconce: Document Once, Include Anywhere Documentation

Release 1.0

Hans Petter Langtangen

July 25, 2012

CONTENTS

Contents:

CHAPTER

ONE

DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

Author Hans Petter Langtangen

Date Jul 25, 2012

- 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.

CHAPTER

TWO

THE DOCONCE CONCEPT

Doconce is two things:

1. Doconce is a very simple and minimally tagged markup language that looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reST you can (via `rst2*` programs) go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter (via `unoconv`) to RTF, numerous MS Word formats (including MS Office Open XML), DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU Texinfo, MediaWiki, RTF, Groff, and other formats.
2. Doconce is a working strategy for never duplicating information. Text is written in a single place and then transformed to a number of different destinations of diverse type (software source code, manuals, tutorials, books, wikis, memos, emails, etc.). The Doconce markup language support this working strategy. The slogan is: "Write once, include anywhere".

Here are some Doconce features:

- Doconce markup does include tags, so the format is more tagged than Markdown and Pandoc, but less than reST, and very much less than LaTeX and HTML.
- Doconce can be converted to plain *untagged* text, often desirable for computer programs and email.
- Doconce has good support for copying in parts of computer code directly from the source code files via regular expressions for the start and end lines.
- Doconce has full support for LaTeX math and integrates well with big LaTeX projects (books).
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
- Contrary to the similar (and superior) Pandoc translator, Doconce supports Sphinx, Google wiki, Creole wiki (for bitbucket.org), lots of computer code environments in LaTeX, and a special exercise syntax. Doconce also also runs preprocessors (including Mako) such that the author can mix ordinary text with programming construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

- Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
- Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at web sites, and as LaTeX integrated in, e.g., a thesis.

- Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover, Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting and handling text through regular expressions. The possibility for tweaking the layout is obviously limited since the text can go to all sorts of sophisticated markup languages. Moreover, because of limitations of regular expressions, some formatting of Doconce syntax may face problems when transformed to HTML, LaTeX, Sphinx, and similar formats.

CHAPTER

THREE

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 formating. Here are som examples.

- Bullet lists arise from lines starting with an asterisk.
- *Emphasized words* are surrounded by asterisks.
- **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 recognied 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 special support for advanced exercises features.
- With a simple 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 the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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 an 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 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 supperted, 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:

3.1 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 supperted, e.g.,

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

3.2 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

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

The pipe symbol acts as a delimiter between LaTeX code and the plain text version of the formula.

Blocks of mathematics are better typeset with raw LaTeX, inside `!bt` and `!et` (begin tex / end tex) instructions. The result looks like this:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla^2 u + f, \\ \frac{\partial v}{\partial t} &= \nabla \cdot (q(u)\nabla v) + g \end{aligned} \tag{3.1}$$

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other 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. Such blocks look like

```
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.

It is possible to add a specification of an environment for typesetting the verbatim code block, e.g., `!bc xxx` where `xxx` is an identifier like `pycod` for code snippet in Python, `sys` for terminal session, etc. When Doconce is filtered to LaTeX, these identifiers are used as in `ptex2tex` and defined in a configuration file `.ptext2tex.cfg`, while when filtering to Sphinx, one can have a comment line in the Doconce file for mapping the identifiers to legal language names for Sphinx (which equals the legal language names for Pygments):

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

By default, `pro` and `cod` are `python`, `sys` is `console`, while `xpro` and `xcod` are computer language specific for `x` in `f` (Fortran), `c` (C), `cpp` (C++), `pl` (Perl), `m` (Matlab), `sh` (Unix shells), `cy` (Cython), and `py` (Python).

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!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. 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 the right writing enviroment for plain text.

3.3 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).

CHAPTER

FOUR

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
```

The `mako` or `preprocess` programs are always used to preprocess the file first, and options to `mako` or `preprocess` 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`. 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"`.

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.

4.1 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.

4.2 Pandoc

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 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. See the Pandoc documentation for the many features of the `pandoc` program.

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 `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> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

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

Quite some `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 makeing HTML notes with (some) mathematics.

4.3 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.

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`.

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 cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}`). 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` or `fpro` are defined, 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. Here are some examples:

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
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 encourged to make a script for generating PDF from the LaTeX file.

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 use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code (specified, e.g., in the `*pro` and `*cod` environments in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the `minted` LaTeX package is needed. This package is included by running `doconce format` 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
```

4.4 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
```

4.5 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
```

4.6 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

4.7 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. You should at least read the script prior to executing it to have some idea of what is done.

The `doconce sphinx_dir` script copies directories named `figs` or `figures` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. If figures or movies are located in other directories, `automake-sphinx.py` must be edited accordingly. Files, to which there are local links (not `http:` or `file:` URLs), must be placed in the `_static` subdirectory of the Sphinx directory. The utility `doconce sphinxfix_localURLs` is run to check for local links in the Doconce file: for each such link, say `dir1/dir2/myfile.txt` it replaces the link by `_static/myfile.txt` and copies `dir1/dir2/myfile.txt` to a local `_static` directory (in the same directory as the script is run). However, we recommend instead that the writer of the document places files in `_static` or lets a script do it automatically. The user must copy all `_static/*` files to the `_static` subdirectory of the Sphinx directory. It may be wise to always put files, to which there are local links in the Doconce document, in a `_static` or `_static-name` directory and use these local links. Then links do not need to be modified when creating a Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents. These are packed out in the Sphinx directory, the `conf.py` configuration file for Sphinx is edited accordingly, and a script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics` and `pyramid`, one writes

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

The resulting directories with HTML documents are _build/html_fenics and _build/html_pyramid, respectively. Without arguments, make-themes.sh makes all available themes (!).

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.

4.8 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 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).

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. At the same time, the book can also be published as a standard LaTeX book or a Sphinx web document.

4.9 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.

4.10 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.

4.11 Dependencies and Installation

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
```

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.

To make LaTeX documents (without going through the reStructuredText format) you need ptex2tex, which is installed by

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
```

As seen, `cp2texmf.sh` 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-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).

The *minted* LaTeX style is offered by `ptex2tex` and popular among users. This style requires the package Pygments:

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

If you use the minted style, you have to enable it by running `ptex2tex -DMINTED` and then `latex -shell-escape`, see the the section *From Doconce to Other Formats*.

For `rst` output and further transformation to LaTeX, HTML, XML, OpenOffice, and so on, one needs docutils. The installation 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 `sudo python setup.py install`.

Output to `sphinx` requires of course Sphinx, installed by

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

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 ..
```

Finally, translation to `pandoc` requires the Pandoc program (written in Haskell) to be installed.

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

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 itself is frequently updated so these commands should be run regularly.

CHAPTER

FIVE

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: Jul 25, 2012

 * 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://do
cutils.sourceforge.net/docs/ref/rst/restructuredtext.html), Sphinx (http://sphin
x.pocoo.org/contents.html), and wiki (http://code.google.com/p/support/wiki/Wiki
Syntax)? 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.


The Doconce Concept
===================

Doconce is two things:

 1. Doconce is a very simple and minimally tagged markup language that
    looks like ordinary ASCII text (much like what you would use in an
    email), but the text can be transformed to numerous other formats,
    including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText
    (reST), Sphinx, Epytext, and also plain text (where non-obvious
    formatting/tags are removed for clear reading in, e.g.,
    emails). From reST you can (via rst2* programs) go to XML, HTML,
    LaTeX, PDF, OpenOffice, and from the latter (via unoconv) to
    RTF, numerous MS Word formats (including MS Office Open XML),
    DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate
    Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU
    Texinfo, MediaWiki, RTF, Groff, and other formats.

 2. Doconce is a working strategy for never duplicating information.
    Text is written in a single place and then transformed to
    a number of different destinations of diverse type (software
    source code, manuals, tutorials, books, wikis, memos, emails, etc.).
```

tutorial.txt

```
    The Doconce markup language support this working strategy.
    The slogan is: "Write once, include anywhere".

Here are some Doconce features:

  * Doconce markup does include tags, so the format is more tagged than
    Markdown and Pandoc, but less than reST, and very much less than
    LaTeX and HTML.

  * Doconce can be converted to plain *untagged* text,
    often desirable for computer programs and email.

  * Doconce has good support for copying in parts of computer code
    directly from the source code files via regular expressions
    for the start and end lines.

  * Doconce has full support for LaTeX math and integrates well
    with big LaTeX projects (books).

  * Doconce is almost self-explanatory and is a handy starting point
    for generating documents in more complicated markup languages, such
    as Google wiki, LaTeX, and Sphinx. A primary application of Doconce
    is just to make the initial versions of a Sphinx or wiki document.

  * Contrary to the similar (and superior) Pandoc translator, Doconce
    supports Sphinx, Google wiki, Creole wiki (for bitbucket.org),
    lots of computer code environments in LaTeX, and a special exercise
    syntax. Doconce also also runs preprocessors (including Mako)
    such that the author can mix ordinary text with programming
    construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

  * Large books written in LaTeX, but where many pieces (computer demos,
    projects, examples) can be written in Doconce to appear in other
    contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.

  * Software documentation, primarily Python doc strings, which one wants
    to appear as plain untagged text for viewing in Pydoc, as reStructuredText
    for use with Sphinx, as wiki text when publishing the software at
    web sites, and as LaTeX integrated in, e.g., a thesis.

  * Quick memos, which start as plain text in email, then some small
    amount of Doconce tagging is added, before the memos can appear as
    Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular
markup languages used quite some tagging.  Later, almost untagged
markup languages like Markdown and Pandoc became popular. Doconce is
not a replacement of Pandoc, which is a considerably more
sophisticated project. Moreover, Doconce was developed mainly to
fulfill the needs for a flexible source code base for books with much
mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting
and handling text through regular expressions. The possibility for
tweaking the layout is obviously limited since the text can go to
all sorts of sophisticated markup languages. Moreover, because of
limitations of regular expressions, some formatting of Doconce syntax
```

```
may face problems when transformed to HTML, LaTeX, Sphinx, and similar
formats.



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 formating. Here are
som examples.

  * Bullet lists arise from lines starting with an asterisk.

  * *Emphasized words* are surrounded by asterisks.

  * _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 recognied 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 special support for advanced exercises features.

  * With a simple preprocessor, Preprocess or Mako, one can include
    other documents (files) and large portions of text can be defined
    in or out of the text.
```

tutorial.txt

```
  * With the Mako preprocessor one can even embed Python
    code and use this to steer generation of Doconce text.

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 an 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 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 supperted, 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:

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,
```

tutorial.txt

```
  * 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
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 supperted, 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.

Blocks of mathematics are better typeset with raw LaTeX, inside
  !bt and !et (begin tex / end tex) instructions.
The result looks like this::

        \begin{eqnarray}
```

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

Of course, such blocks only looks nice in LaTeX. The raw
LaTeX syntax appears in all other 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. Such blocks look like::


        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.

It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., !bc xxx
where xxx is an identifier like pycod for code snippet in Python,
sys for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in ptex2tex and defined in a
configuration file .ptext2tex.cfg, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments)::


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

By default, pro and cod are python, sys is console,
while xpro and xcod are computer language specific for x
in f (Fortran), c (C), cpp (C++), pl (Perl), m (Matlab),
sh (Unix shells), cy (Cython), and py (Python).


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!). A complete file is typeset
with !bc pro, while a part of a file is copied into a !bc cod
environment. What pro and cod mean is then defined through
a .ptex2tex.cfg file for LaTeX and a sphinx code-blocks
comment for Sphinx.

Another document can be included by writing #include "mynote.do.txt"
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.


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

tutorial.txt

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

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 (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

The mako or preprocess programs are always used to preprocess the
file first, and options to mako or preprocess 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. 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".

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.


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.

Pandoc
------

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 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.
See the Pandoc documentation (http://johnmacfarlane.net/pandoc/README.html)
for the many features of the pandoc program.

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 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> pandoc -f latex -t docx -o mydoc.docx mydoc.tex

When we go through pandoc, only single equations or align*
environments are well understood.

Quite some 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 makeing HTML notes with (some) mathematics.


LaTeX
-----

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.

*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.

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 cod for a code snippet, where cod is set to
a certain environment in .ptex2tex.cfg (e.g., CodeIntended).
There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
                  "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}"
\
                  fpro=minted fcod=minted 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}).
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 or fpro are
defined, 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.
Here are some examples::


        Terminal> doconce replace 'section{' 'section*{' mydoc.tex
        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 encourged to make a script for
generating PDF from the LaTeX file.

*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 use the Minted_Python, Minted_Cpp, etc.,
environments in ptex2tex for typesetting code (specified, e.g., in
the *pro and *cod environments in .ptex2tex.cfg or
$HOME/.ptex2tex.cfg), the minted LaTeX package is needed.  This
package is included by running doconce format 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



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 n
o 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.
You should at least read the script prior to executing it to have
some idea of what is done.

The doconce sphinx_dir script copies directories named figs or
figures over to the Sphinx directory so that figures are accessible
in the Sphinx compilation.  If figures or movies are located in other
directories, automake-sphinx.py must be edited accordingly.  Files,
to which there are local links (not http: or file: URLs), must be
placed in the _static subdirectory of the Sphinx directory. The
utility doconce sphinxfix_localURLs is run to check for local links
in the Doconce file: for each such link, say dir1/dir2/myfile.txt it
replaces the link by _static/myfile.txt and copies
dir1/dir2/myfile.txt to a local _static directory (in the same
directory as the script is run).  However, we recommend instead that
the writer of the document places files in _static or lets a script
do it automatically. The user must copy all _static/* files to the
_static subdirectory of the Sphinx directory.  It may be wise to
always put files, to which there are local links in the Doconce
document, in a _static or _static-name directory and use these
local links. Then links do not need to be modified when creating a
Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents.
These are packed out in the Sphinx directory, the conf.py
configuration file for Sphinx is edited accordingly, and a script
make-themes.sh can make HTML documents with one or more themes.
For example,
to realize the themes fenics and pyramid, one writes::
```

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

The resulting directories with HTML documents are _build/html_fenics
and _build/html_pyramid, respectively. Without arguments,
make-themes.sh makes all available themes (!).

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 (http://code.google.com/p/support/wiki/WikiSyntax), MediaWiki, a
nd 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 Googlecode wiki document, mydoc.gwiki, is most conveniently stored
in a directory which is a clone of the wiki part of the Googlecode project.
```

tutorial.txt

```
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).

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.
At the same time, the book can also be published as a
standard LaTeX book or a Sphinx web document.


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.

Dependencies and Installation
-----------------------------

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://doconce.googlecode.com/hg/ doconce
        cd doconce
        sudo python setup.py install
        cd ..


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

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).

To make LaTeX
documents (without going through the reStructuredText format) you
need 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
        cd latex
        sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
        cd ../..

As seen, cp2texmf.sh 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-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).
```

tutorial.txt

```
The *minted* LaTeX style is offered by ptex2tex and popular among
users. This style requires the package Pygments (http://pygments.org)::


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

If you use the minted style, you have to enable it by running
ptex2tex -DMINTED and then latex -shell-escape, see
the the section "From Doconce to Other Formats".

For rst output and further transformation to LaTeX, HTML, XML,
OpenOffice, and so on, one needs docutils (http://docutils.sourceforge.net).
The installation 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 (http://code.google.com/p/rst2pdf). Either download the tarball
or clone the svn repository, go to the rst2pdf directory and
run sudo python setup.py install.


Output to sphinx requires of course Sphinx (http://sphinx.pocoo.org),
installed by::


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


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


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


Finally, translation to pandoc requires the
Pandoc (http://johnmacfarlane.net/pandoc/) program
```

```
(written in Haskell) to be installed::

        sudo apt-get install pandoc

*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 itself is frequently updated so these commands should be
run regularly.
```

tutorial.epytext

```
TITLE: Doconce: Document Once, Include Anywhere
BY: Hans Petter Langtangen (Simula Research Laboratory, and University of Oslo)D
ATE: 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 U{LaTeX<http://refcards.com/docs/silvermanj/amslatex/LaTeXRefCar
d.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.


The Doconce Concept
==================

Doconce is two things:

 1. Doconce is a very simple and minimally tagged markup language that
    looks like ordinary ASCII text (much like what you would use in an
    email), but the text can be transformed to numerous other formats,
    including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText
    (reST), Sphinx, Epytext, and also plain text (where non−obvious
    formatting/tags are removed for clear reading in, e.g.,
    emails). From reST you can (via C{rst2*} programs) go to XML, HTML,
    LaTeX, PDF, OpenOffice, and from the latter (via C{unoconv}) to
    RTF, numerous MS Word formats (including MS Office Open XML),
    DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate
    Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU
    Texinfo, MediaWiki, RTF, Groff, and other formats.
 2. Doconce is a working strategy for never duplicating information.
    Text is written in a single place and then transformed to
    a number of different destinations of diverse type (software
    source code, manuals, tutorials, books, wikis, memos, emails, etc.).
    The Doconce markup language support this working strategy.
    The slogan is: "Write once, include anywhere".

Here are some Doconce features:

  − Doconce markup does include tags, so the format is more tagged than
    Markdown and Pandoc, but less than reST, and very much less than
    LaTeX and HTML.
  − Doconce can be converted to plain I{untagged} text,
```

```
    often desirable for computer programs and email.
  - Doconce has good support for copying in parts of computer code
    directly from the source code files via regular expressions
    for the start and end lines.
  - Doconce has full support for LaTeX math and integrates well
    with big LaTeX projects (books).
  - Doconce is almost self-explanatory and is a handy starting point
    for generating documents in more complicated markup languages, such
    as Google wiki, LaTeX, and Sphinx. A primary application of Doconce
    is just to make the initial versions of a Sphinx or wiki document.
  - Contrary to the similar (and superior) Pandoc translator, Doconce
    supports Sphinx, Google wiki, Creole wiki (for bitbucket.org),
    lots of computer code environments in LaTeX, and a special exercise
    syntax. Doconce also also runs preprocessors (including Mako)
    such that the author can mix ordinary text with programming
    construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

  - Large books written in LaTeX, but where many pieces (computer demos,
    projects, examples) can be written in Doconce to appear in other
    contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
  - Software documentation, primarily Python doc strings, which one wants
    to appear as plain untagged text for viewing in Pydoc, as reStructuredText
    for use with Sphinx, as wiki text when publishing the software at
    web sites, and as LaTeX integrated in, e.g., a thesis.
  - Quick memos, which start as plain text in email, then some small
    amount of Doconce tagging is added, before the memos can appear as
    Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular
markup languages used quite some tagging.  Later, almost untagged
markup languages like Markdown and Pandoc became popular. Doconce is
not a replacement of Pandoc, which is a considerably more
sophisticated project. Moreover, Doconce was developed mainly to
fulfill the needs for a flexible source code base for books with much
mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting
and handling text through regular expressions. The possibility for
tweaking the layout is obviously limited since the text can go to
all sorts of sophisticated markup languages. Moreover, because of
limitations of regular expressions, some formatting of Doconce syntax
may face problems when transformed to HTML, LaTeX, Sphinx, and similar
formats.



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 formating. Here are
som examples.

  - Bullet lists arise from lines starting with an asterisk.
  - I{Emphasized words} are surrounded by asterisks.
  - 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 recognied 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
    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 special support for advanced exercises features.
  - With a simple 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 the Mako preprocessor one can even embed Python
    code and use this to steer generation of Doconce text.

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 an 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.epytext

```
        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 supperted, 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:

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 supperted, 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.


The pipe symbol acts as a delimiter between LaTeX code and the plain text
version of the formula.

Blocks of mathematics are better typeset with raw LaTeX, inside
C{!bt} and C{!et} (begin tex / end tex) instructions.
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 LaTeX. The raw
LaTeX syntax appears in all other 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. Such blocks look like::


        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 C{sphinx}, C{rst}, and ASCII-close formats),
not directly after a section/paragraph heading or a table.

It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., C{!bc xxx}
where C{xxx} is an identifier like C{pycod} for code snippet in Python,
C{sys} for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in C{ptex2tex} and defined in a
configuration file C{.ptext2tex.cfg}, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments)::
```

tutorial.epytext

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

By default, C{pro} and C{cod} are C{python}, C{sys} is C{console},
while C{xpro} and C{xcod} are computer language specific for C{x}
in C{f} (Fortran), C{c} (C), C{cpp} (C++), C{pl} (Perl), C{m} (Matlab),
C{sh} (Unix shells), C{cy} (Cython), and C{py} (Python).


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!). A complete file is typeset
with C{!bc pro}, while a part of a file is copied into a C{!bc cod}
environment. What C{pro} and C{cod} mean is then defined through
a C{.ptex2tex.cfg} file for LaTeX and a C{sphinx code-blocks}
comment for Sphinx.

Another document can be included by writing C{#include "mynote.do.txt"}
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.


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
=============================

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

The C{mako} or C{preprocess} programs are always used to preprocess the
file first, and options to C{mako} or C{preprocess} 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}. 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"}.

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.


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.
```

tutorial.epytext

```
Pandoc
------

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 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.
See the U{Pandoc documentation<http://johnmacfarlane.net/pandoc/README.html>}
for the many features of the C{pandoc} program.

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{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
        Terminal> doconce ptex2tex mydoc
        Terminal> pandoc -f latex -t docx -o mydoc.docx mydoc.tex

When we go through C{pandoc}, only single equations or C{align*}
environments are well understood.

Quite some 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 makeing 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-Referen
ces, Index, and Bibliography").
If these files are present, they are included in the LaTeX document
so that your commands are defined.

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


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}.

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 cod} for a code snippet, where C{cod} is set to
a certain environment in C{.ptex2tex.cfg} (e.g., C{CodeIntended}).
There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
                  "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}"
\
                  fpro=minted fcod=minted 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}}).
```

```
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} or C{fpro} are
defined, 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.
Here are some examples::


        Terminal> doconce replace 'section{' 'section*{' mydoc.tex
        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 encourged to make a script for
generating PDF from the LaTeX file.

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 use the C{Minted_Python}, C{Minted_Cpp}, etc.,
environments in C{ptex2tex} for typesetting code (specified, e.g., in
the C{*pro} and C{*cod} environments in C{.ptex2tex.cfg} or
C{$HOME/.ptex2tex.cfg}), the C{minted} LaTeX package is needed.  This
package is included by running C{doconce format} with the C{-DMINTED}
option::


        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
```

tutorial.epytext

```
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

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/sh
owthread.php?t=1033441>}
 - U{http://tug.org/utilities/texconv/textopc.html<http://tug.org/utilities/texc
onv/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
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.
You should at least read the script prior to executing it to have
some idea of what is done.
```

tutorial.epytext

```
The C{doconce sphinx_dir} script copies directories named C{figs} or
C{figures} over to the Sphinx directory so that figures are accessible
in the Sphinx compilation.  If figures or movies are located in other
directories, C{automake-sphinx.py} must be edited accordingly.  Files,
to which there are local links (not C{http:} or C{file:} URLs), must be
placed in the C{_static} subdirectory of the Sphinx directory. The
utility C{doconce sphinxfix_localURLs} is run to check for local links
in the Doconce file: for each such link, say C{dir1/dir2/myfile.txt} it
replaces the link by C{_static/myfile.txt} and copies
C{dir1/dir2/myfile.txt} to a local C{_static} directory (in the same
directory as the script is run).  However, we recommend instead that
the writer of the document places files in C{_static} or lets a script
do it automatically. The user must copy all C{_static/*} files to the
C{_static} subdirectory of the Sphinx directory.  It may be wise to
always put files, to which there are local links in the Doconce
document, in a C{_static} or C{_static-name} directory and use these
local links. Then links do not need to be modified when creating a
Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents.
These are packed out in the Sphinx directory, the C{conf.py}
configuration file for Sphinx is edited accordingly, and a script
C{make-themes.sh} can make HTML documents with one or more themes.
For example,
to realize the themes C{fenics} and C{pyramid}, one writes::


        Terminal> ./make-themes.sh fenics pyramid

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 (!).

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}
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>}, MediaWiki,
 and Creole Wiki. 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 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).

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.
At the same time, the book can also be published as a
standard LaTeX book or a Sphinx web document.


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.
```

tutorial.epytext

```
Demos
-----

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


        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.

Dependencies and Installation
-----------------------------

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://doconce.googlecode.com/hg/ doconce
        cd doconce
        sudo python setup.py install
        cd ..


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
```

tutorial.epytext

```
Alternatively, one can install from the C{pip} U{source code<http://pypi.python.
org/pypi/pip>}.

To make LaTeX
documents (without going through the reStructuredText format) 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
        cd latex
        sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
        cd ../..

As seen, C{cp2texmf.sh} 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-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).

The I{minted} LaTeX style is offered by C{ptex2tex} and popular among
users. This style requires the package U{Pygments<http://pygments.org>}::


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

If you use the minted style, you have to enable it by running
C{ptex2tex -DMINTED} and then C{latex -shell-escape}, see
the the section "From Doconce to Other Formats".

For C{rst} output and further transformation to LaTeX, HTML, XML,
OpenOffice, and so on, one needs U{docutils<http://docutils.sourceforge.net>}.
The installation 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 C{sudo python setup.py install}.


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


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


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


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


Finally, translation to C{pandoc} requires the
U{Pandoc<http://johnmacfarlane.net/pandoc/>} program
(written in Haskell) to be installed::


        sudo apt-get install pandoc


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}.
Doconce itself is frequently updated so these commands should be
run regularly.
```

tutorial.gwiki

```
#summary Doconce: Document Once, Include Anywhere
<wiki:toc max_depth="2" />
By *Hans Petter Langtangen*

==== Jul 25, 2012 ====

 * When writing a note, report, manual, etc., do you find it difficult   to choo
se the typesetting format? That is, to choose between plain   (email-like) text,
 wiki, Word/OpenOffice, LaTeX, HTML,   reStructuredText, Sphinx, XML, etc.  Woul
d 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   eventual
ly 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.htmlcodet
utorial.com/ HTML], [http://docutils.sourceforge.net/docs/ref/rst/restructuredte
xt.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.



== The Doconce Concept ==

Doconce is two things:


 # Doconce is a very simple and minimally tagged markup language that    looks l
ike ordinary ASCII text (much like what you would use in an    email), but the t
ext can be transformed to numerous other formats,    including HTML, Pandoc, Goo
gle wiki, LaTeX, PDF, reStructuredText    (reST), Sphinx, Epytext, and also plai
n text (where non-obvious    formatting/tags are removed for clear reading in, e
.g.,    emails). From reST you can (via 'rst2*' programs) go to XML, HTML,    La
TeX, PDF, OpenOffice, and from the latter (via 'unoconv') to    RTF, numerous MS
 Word formats (including MS Office Open XML),    DocBook, PDF, MediaWiki, XHTML.
 From Pandoc one can generate    Markdown, reST, LaTeX, HTML, PDF, DocBook XML,
OpenOffice, GNU    Texinfo, MediaWiki, RTF, Groff, and other formats.
 # Doconce is a working strategy for never duplicating information.    Text is w
ritten in a single place and then transformed to    a number of different destin
ations of diverse type (software    source code, manuals, tutorials, books, wiki
s, memos, emails, etc.).    The Doconce markup language support this working str
ategy.    The slogan is: "Write once, include anywhere".

Here are some Doconce features:


  * Doconce markup does include tags, so the format is more tagged than    Markd
own and Pandoc, but less than reST, and very much less than    LaTeX and HTML.
  * Doconce can be converted to plain *untagged* text,    often desirable for co
mputer programs and email.
  * Doconce has good support for copying in parts of computer code    directly f
rom the source code files via regular expressions    for the start and end lines
.
```

* Doconce has full support for LaTeX math and integrates well with big LaTeX projects (books).
* Doconce is almost self−explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
* Contrary to the similar (and superior) Pandoc translator, Doconce supports Sphinx, Google wiki, Creole wiki (for bitbucket.org), lots of computer code environments in LaTeX, and a special exercise syntax. Doconce also also runs preprocessors (including Mako) such that the author can mix ordinary text with programming construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

* Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
* Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at web sites, and as LaTeX integrated in, e.g., a thesis.
* Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover, Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting and handling text through regular expressions. The possibility for tweaking the layout is obviously limited since the text can go to all sorts of sophisticated markup languages. Moreover, because of limitations of regular expressions, some formatting of Doconce syntax may face problems when transformed to HTML, LaTeX, Sphinx, and similar formats.

== 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 formating. Here are som examples.

* Bullet lists arise from lines starting with an asterisk.
* *Emphasized words* are surrounded by asterisks.
* *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 recognied 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, a
nd all text up to the next heading makes up the abstract,
  * Blocks of computer code can easily be included by placing    ‘!bc‘ (begin co
de) 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‘ (begi
n TeX) and ‘!et‘ (end TeX) commands at separate lines    before and after the ma
th block.
  * There is support for both LaTeX and text-like inline mathematics.
  * Figures and movies with captions, simple tables,    URLs with links, index l
ist, labels and references are supported.
  * Invisible comments in the output format can be inserted throughout    the te
xt.
  * 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 special support for advanced exercises features.
  * With a simple preprocessor, Preprocess or Mako, one can include    other doc
uments (files) and large portions of text can be defined    in or out of the tex
t.
  * With the Mako preprocessor one can even embed Python    code and use this to
 steer generation of Doconce text.

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 an 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 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 supperted, 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:

```
==== 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:


 # 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 supperted, 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
```

tutorial.gwiki

```
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.

Blocks of mathematics are better typeset with raw LaTeX, inside
‘!bt‘ and ‘!et‘ (begin tex / end tex) instructions.
The result looks like this:
{{{
\begin{eqnarray}
{\partial u\over\partial t} &=& \nabla^2 u + f, label{myeq1}\\
{\partial v\over\partial t} &=& \nabla\cdot(q(u)\nabla v) + g
\end{eqnarray}
}}}
Of course, such blocks only looks nice in LaTeX. The raw
LaTeX syntax appears in all other 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. Such blocks look like
{{{
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.

It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., ‘!bc xxx‘
where ‘xxx‘ is an identifier like ‘pycod‘ for code snippet in Python,
‘sys‘ for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in ‘ptex2tex‘ and defined in a
configuration file ‘.ptext2tex.cfg‘, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):
{{{
# sphinx code-blocks: pycod=python cod=fortran cppcod=c++ sys=console
}}}
By default, ‘pro‘ and ‘cod‘ are ‘python‘, ‘sys‘ is ‘console‘,
while ‘xpro‘ and ‘xcod‘ are computer language specific for ‘x‘
in ‘f‘ (Fortran), ‘c‘ (C), ‘cpp‘ (C++), ‘pl‘ (Perl), ‘m‘ (Matlab),
‘sh‘ (Unix shells), ‘cy‘ (Cython), and ‘py‘ (Python).

<wiki:comment> (Any sphinx code-block comment, whether inside verbatim code </wi
ki:comment>
<wiki:comment> blocks or outside, yields a mapping between bc arguments </wiki:c
omment>
<wiki:comment> and computer languages. In case of muliple definitions, the </wik
i:comment>
<wiki:comment> first one is used.) </wiki:comment>
```

```
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!). A complete file is typeset
with ‘!bc pro‘, while a part of a file is copied into a ‘!bc cod‘
environment. What ‘pro‘ and ‘cod‘ mean is then defined through
a ‘.ptex2tex.cfg‘ file for LaTeX and a ‘sphinx code-blocks‘
comment for Sphinx.

Another document can be included by writing ‘#include "mynote.do.txt"‘
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.

==== 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): </w
iki:comment>



== From Doconce to Other Formats ==

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

tutorial.gwiki

```
{{{
Terminal> doconce format format mydoc.do.txt
}}}
or just
{{{
Terminal> doconce format format mydoc
}}}
The ‘mako‘ or ‘preprocess‘ programs are always used to preprocess the
file first, and options to ‘mako‘ or ‘preprocess‘ 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‘. 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"‘.

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.

==== 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.

==== Pandoc ====

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 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.
See the [http://johnmacfarlane.net/pandoc/README.html Pandoc documentation]
for the many features of the ‘pandoc‘ program.

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 ‘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> pandoc −f latex −t docx −o mydoc.docx mydoc.tex
}}}
When we go through ‘pandoc‘, only single equations or ‘align*‘
environments are well understood.

Quite some ‘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 makeing 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 paragra
ph </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−Referen
ces,_Index,_and_Bibliography]).
If these files are present, they are included in the LaTeX document
so that your commands are defined.

*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‘.

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 cod‘ for a code snippet, where ‘cod‘ is set to
a certain environment in ‘.ptex2tex.cfg‘ (e.g., ‘CodeIntended‘).
There are about 40 styles to choose from.

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 −DLATEX_HEADING=traditional −DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}‘).
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‘ or ‘fpro‘ are
defined, 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.
Here are some examples:
{{{
Terminal> doconce replace ’section{’ ’section*{’ mydoc.tex
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 encourged to make a script for
generating PDF from the LaTeX file.

*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 use the ‘Minted_Python‘, ‘Minted_Cpp‘, etc.,
environments in ‘ptex2tex‘ for typesetting code (specified, e.g., in
the ‘*pro‘ and ‘*cod‘ environments in ‘.ptex2tex.cfg‘ or
‘$HOME/.ptex2tex.cfg‘), the ‘minted‘ LaTeX package is needed.  This
package is included by running ‘doconce format‘ 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
}}}

==== 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 n
o 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.
You should at least read the script prior to executing it to have
some idea of what is done.

The ‘doconce sphinx_dir‘ script copies directories named ‘figs‘ or
‘figures‘ over to the Sphinx directory so that figures are accessible
in the Sphinx compilation.  If figures or movies are located in other
directories, ‘automake-sphinx.py‘ must be edited accordingly.  Files,
to which there are local links (not ‘http:‘ or ‘file:‘ URLs), must be
placed in the ‘_static‘ subdirectory of the Sphinx directory. The
utility ‘doconce sphinxfix_localURLs‘ is run to check for local links
in the Doconce file: for each such link, say ‘dir1/dir2/myfile.txt‘ it
replaces the link by ‘_static/myfile.txt‘ and copies
‘dir1/dir2/myfile.txt‘ to a local ‘_static‘ directory (in the same
directory as the script is run).  However, we recommend instead that
the writer of the document places files in ‘_static‘ or lets a script
do it automatically. The user must copy all ‘_static/*‘ files to the
‘_static‘ subdirectory of the Sphinx directory.  It may be wise to
always put files, to which there are local links in the Doconce
document, in a ‘_static‘ or ‘_static-name‘ directory and use these
local links. Then links do not need to be modified when creating a
Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents.
These are packed out in the Sphinx directory, the ‘conf.py‘
configuration file for Sphinx is edited accordingly, and a script
‘make-themes.sh‘ can make HTML documents with one or more themes.
For example,
to realize the themes ‘fenics‘ and ‘pyramid‘, one writes
{{{
Terminal> ./make-themes.sh fenics pyramid
}}}
The resulting directories with HTML documents are ‘_build/html_fenics‘
and ‘_build/html_pyramid‘, respectively. Without arguments,
‘make-themes.sh‘ makes all available themes (!).

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:
[http://code.google.com/p/support/wiki/WikiSyntax Googlecode wiki], MediaWiki, a
nd 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 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).

From the MediaWiki format one can go to other formats with aid
of [http://pediapress.com/code/ mwlib]. This means that one can
easily use Doconce to write [http://en.wikibooks.org Wikibooks]
and publish these in PDF and MediaWiki format.
At the same time, the book can also be published as a
standard LaTeX book or a Sphinx web document.

==== 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.

==== Dependencies and Installation ====

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
}}}

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 s
ource code].

To make LaTeX
documents (without going through the reStructuredText format) 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
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
}}}
As seen, ‘cp2texmf.sh‘ 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-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).

The *minted* LaTeX style is offered by ‘ptex2tex‘ and popular among
users. This style requires the package [http://pygments.org Pygments]:
{{{
hg clone ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments
cd pygments
sudo python setup.py install
}}}
If you use the minted style, you have to enable it by running
‘ptex2tex -DMINTED‘ and then ‘latex -shell-escape‘, see
the the section [#From_Doconce_to_Other_Formats].

For ‘rst‘ output and further transformation to LaTeX, HTML, XML,
OpenOffice, and so on, one needs [http://docutils.sourceforge.net docutils].
The installation 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 ‘sudo python setup.py install‘.


Output to ‘sphinx‘ requires of course [http://sphinx.pocoo.org Sphinx],
installed by
{{{
hg clone https://bitbucket.org/birkenfeld/sphinx
cd sphinx
sudo python setup.py install
cd ..
}}}

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 ..
}}}

Finally, translation to ‘pandoc‘ requires the
[http://johnmacfarlane.net/pandoc/ Pandoc] program
(written in Haskell) to be installed.
{{{
sudo apt-get install pandoc
}}}

*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 itself is frequently updated so these commands should be
run regularly.
```

tutorial.mkd

```
% Doconce: Document Once, Include Anywhere
% Hans Petter Langtangen at Simula Research Laboratory and University of Oslo
% Jul 25, 2012

 * 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/LaTeXRefCar
d.v2.0.pdf), [HTML](http://www.htmlcodetutorial.com/), [reStructuredText](http:/
/docutils.sourceforge.net/docs/ref/rst/restructuredtext.html), [Sphinx](http://s
phinx.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.


The Doconce Concept
===================

Doconce is two things:

 1. Doconce is a very simple and minimally tagged markup language that
    looks like ordinary ASCII text (much like what you would use in an
    email), but the text can be transformed to numerous other formats,
    including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText
    (reST), Sphinx, Epytext, and also plain text (where non-obvious
    formatting/tags are removed for clear reading in, e.g.,
    emails). From reST you can (via `rst2*` programs) go to XML, HTML,
    LaTeX, PDF, OpenOffice, and from the latter (via `unoconv`) to
    RTF, numerous MS Word formats (including MS Office Open XML),
    DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate
    Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU
    Texinfo, MediaWiki, RTF, Groff, and other formats.

 2. Doconce is a working strategy for never duplicating information.
    Text is written in a single place and then transformed to
    a number of different destinations of diverse type (software
    source code, manuals, tutorials, books, wikis, memos, emails, etc.).
    The Doconce markup language support this working strategy.
    The slogan is: "Write once, include anywhere".

Here are some Doconce features:

  * Doconce markup does include tags, so the format is more tagged than
    Markdown and Pandoc, but less than reST, and very much less than
```

```
    LaTeX and HTML.

  * Doconce can be converted to plain *untagged* text,
    often desirable for computer programs and email.

  * Doconce has good support for copying in parts of computer code
    directly from the source code files via regular expressions
    for the start and end lines.

  * Doconce has full support for LaTeX math and integrates well
    with big LaTeX projects (books).

  * Doconce is almost self-explanatory and is a handy starting point
    for generating documents in more complicated markup languages, such
    as Google wiki, LaTeX, and Sphinx. A primary application of Doconce
    is just to make the initial versions of a Sphinx or wiki document.

  * Contrary to the similar (and superior) Pandoc translator, Doconce
    supports Sphinx, Google wiki, Creole wiki (for bitbucket.org),
    lots of computer code environments in LaTeX, and a special exercise
    syntax. Doconce also also runs preprocessors (including Mako)
    such that the author can mix ordinary text with programming
    construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

  * Large books written in LaTeX, but where many pieces (computer demos,
    projects, examples) can be written in Doconce to appear in other
    contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.

  * Software documentation, primarily Python doc strings, which one wants
    to appear as plain untagged text for viewing in Pydoc, as reStructuredText
    for use with Sphinx, as wiki text when publishing the software at
    web sites, and as LaTeX integrated in, e.g., a thesis.

  * Quick memos, which start as plain text in email, then some small
    amount of Doconce tagging is added, before the memos can appear as
    Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular
markup languages used quite some tagging.  Later, almost untagged
markup languages like Markdown and Pandoc became popular. Doconce is
not a replacement of Pandoc, which is a considerably more
sophisticated project. Moreover, Doconce was developed mainly to
fulfill the needs for a flexible source code base for books with much
mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting
and handling text through regular expressions. The possibility for
tweaking the layout is obviously limited since the text can go to
all sorts of sophisticated markup languages. Moreover, because of
limitations of regular expressions, some formatting of Doconce syntax
may face problems when transformed to HTML, LaTeX, Sphinx, and similar
formats.



What Does Doconce Look Like?
============================
```

tutorial.mkd

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

  * Bullet lists arise from lines starting with an asterisk.

  * *Emphasized words* are surrounded by asterisks.

  * _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 recognied 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 special support for advanced exercises features.

  * With a simple 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 the Mako preprocessor one can even embed Python
    code and use this to steer generation of Doconce text.

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

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

tutorial.mkd

```
===== 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 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 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 supperted, 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:

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 supperted, 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 $\nu = \sin(x)$,
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

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

The pipe symbol acts as a delimiter between LaTeX code and the plain text
version of the formula.

Blocks of mathematics are better typeset with raw LaTeX, inside
‘!bt‘ and ‘!et‘ (begin tex / end tex) instructions.
The result looks like this:
$$
\begin{eqnarray}
{\partial u\over\partial t} &=& \nabla^2 u + f, \label{myeq1}\\
{\partial v\over\partial t} &=& \nabla\cdot(q(u)\nabla v) + g
\end{eqnarray}
$$
Of course, such blocks only looks nice in LaTeX. The raw
LaTeX syntax appears in all other 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. Such blocks look like

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~{.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.

It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., ‘!bc xxx‘
where ‘xxx‘ is an identifier like ‘pycod‘ for code snippet in Python,
‘sys‘ for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in ‘ptex2tex‘ and defined in a
configuration file ‘.ptext2tex.cfg‘, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):

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

By default, ‘pro‘ and ‘cod‘ are ‘python‘, ‘sys‘ is ‘console‘,
while ‘xpro‘ and ‘xcod‘ are computer language specific for ‘x‘
in ‘f‘ (Fortran), ‘c‘ (C), ‘cpp‘ (C++), ‘pl‘ (Perl), ‘m‘ (Matlab),
‘sh‘ (Unix shells), ‘cy‘ (Cython), and ‘py‘ (Python).


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!). A complete file is typeset
with ‘!bc pro‘, while a part of a file is copied into a ‘!bc cod‘
environment. What ‘pro‘ and ‘cod‘ mean is then defined through
a ‘.ptex2tex.cfg‘ file for LaTeX and a ‘sphinx code-blocks‘
comment for Sphinx.

Another document can be included by writing ‘#include "mynote.do.txt"‘
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.


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](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‘:

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

or just

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

The ‘mako‘ or ‘preprocess‘ programs are always used to preprocess the
file first, and options to ‘mako‘ or ‘preprocess‘ 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‘. 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"‘.

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.


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.

Pandoc
------

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 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.
See the [Pandoc documentation](http://johnmacfarlane.net/pandoc/README.html)
for the many features of the ‘pandoc‘ program.

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 ‘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> pandoc −f latex −t docx −o mydoc.docx mydoc.tex
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

When we go through ‘pandoc‘, only single equations or ‘align*‘
environments are well understood.

Quite some ‘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:

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~{.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 makeing HTML notes with (some) mathematics.


LaTeX
−−−−−

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‘):

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~{.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−Referenc
es, Index, and Bibliography](#y)).
If these files are present, they are included in the LaTeX document
so that your commands are defined.

*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
```

tutorial.mkd

~~~~~~

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
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‘.

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 cod‘ for a code snippet, where ‘cod‘ is set to
a certain environment in ‘.ptex2tex.cfg‘ (e.g., ‘CodeIntended‘).
There are about 40 styles to choose from.

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 −DLATEX_HEADING=traditional −DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}‘).
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‘ or ‘fpro‘ are
defined, 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.
Here are some examples:

~~~~~~{.Bash}
Terminal> doconce replace ’section{’ ’section*{’ mydoc.tex
Terminal> doconce subst ’title\{(.+)Using (.+)\}’ \
          ’title{\g<1> \\\\ [1.5mm] Using \g<2>’ mydoc.tex

tutorial.mkd

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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 encourged to make a script for
generating PDF from the LaTeX file.

*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 use the ‘Minted_Python‘, ‘Minted_Cpp‘, etc.,
environments in ‘ptex2tex‘ for typesetting code (specified, e.g., in
the ‘*pro‘ and ‘*cod‘ environments in ‘.ptex2tex.cfg‘ or
‘$HOME/.ptex2tex.cfg‘), the ‘minted‘ LaTeX package is needed.  This
package is included by running ‘doconce format‘ with the ‘-DMINTED‘
option:

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

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
```

tutorial.mkd

~~~~~~

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:

~~~~~~{.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 n
o 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.
~~~~~~

tutorial.mkd

```
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.
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.
You should at least read the script prior to executing it to have
some idea of what is done.

The ‘doconce sphinx_dir‘ script copies directories named ‘figs‘ or
‘figures‘ over to the Sphinx directory so that figures are accessible
in the Sphinx compilation.  If figures or movies are located in other
directories, ‘automake-sphinx.py‘ must be edited accordingly.  Files,
to which there are local links (not ‘http:‘ or ‘file:‘ URLs), must be
placed in the ‘_static‘ subdirectory of the Sphinx directory. The
utility ‘doconce sphinxfix_localURLs‘ is run to check for local links
in the Doconce file: for each such link, say ‘dir1/dir2/myfile.txt‘ it
replaces the link by ‘_static/myfile.txt‘ and copies
‘dir1/dir2/myfile.txt‘ to a local ‘_static‘ directory (in the same
directory as the script is run).  However, we recommend instead that
the writer of the document places files in ‘_static‘ or lets a script
do it automatically. The user must copy all ‘_static/*‘ files to the
```

tutorial.mkd

‘_static‘ subdirectory of the Sphinx directory. It may be wise to
always put files, to which there are local links in the Doconce
document, in a ‘_static‘ or ‘_static-name‘ directory and use these
local links. Then links do not need to be modified when creating a
Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents.
These are packed out in the Sphinx directory, the ‘conf.py‘
configuration file for Sphinx is edited accordingly, and a script
‘make-themes.sh‘ can make HTML documents with one or more themes.
For example,
to realize the themes ‘fenics‘ and ‘pyramid‘, one writes

~~~~~~{.Bash}
Terminal> ./make-themes.sh fenics pyramid
~~~~~~

The resulting directories with HTML documents are ‘_build/html_fenics‘
and ‘_build/html_pyramid‘, respectively. Without arguments,
‘make-themes.sh‘ makes all available themes (!).

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
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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.
```

tutorial.mkd

```
Wiki Formats
------------

There are many different wiki formats, but Doconce only supports three:
[Googlecode wiki](http://code.google.com/p/support/wiki/WikiSyntax), MediaWiki,
and Creole Wiki. 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 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).

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.
At the same time, the book can also be published as a
standard LaTeX book or a Sphinx web document.


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
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

tutorial.mkd

```
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.

Dependencies and Installation
−−−−−−−−−−−−−−−−−−−−−−−−−−−−

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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.o
rg/pypi/pip).

To make LaTeX
documents (without going through the reStructuredText format) you
```

tutorial.mkd

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

As seen, ‘cp2texmf.sh‘ 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-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).

The *minted* LaTeX style is offered by ‘ptex2tex‘ and popular among
users. This style requires the package [Pygments](http://pygments.org):

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~{.Bash}
hg clone ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments
cd pygments
sudo python setup.py install
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you use the minted style, you have to enable it by running
‘ptex2tex -DMINTED‘ and then ‘latex -shell-escape‘, see
the the section [From Doconce to Other Formats](#s).

For ‘rst‘ output and further transformation to LaTeX, HTML, XML,
OpenOffice, and so on, one needs [docutils](http://docutils.sourceforge.net).
The installation 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 ‘sudo python setup.py install‘.
```

````
Output to ‘sphinx‘ requires of course [Sphinx](http://sphinx.pocoo.org),
installed by

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~{.Bash}
hg clone https://bitbucket.org/birkenfeld/sphinx
cd sphinx
sudo python setup.py install
cd ..
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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 ..
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Finally, translation to ‘pandoc‘ requires the
[Pandoc](http://johnmacfarlane.net/pandoc/) program
(written in Haskell) to be installed.

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

*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 itself is frequently updated so these commands should be
run regularly.
````

Doconce: Document Once, Include Anywhere Documentation

Release 1.0

Hans Petter Langtangen

July 25, 2012

CONTENTS

Contents:

CHAPTER

ONE

DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

Author Hans Petter Langtangen

Date Jul 25, 2012

- 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.

CHAPTER

TWO

THE DOCONCE CONCEPT

Doconce is two things:

1. Doconce is a very simple and minimally tagged markup language that looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reST you can (via `rst2*` programs) go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter (via `unoconv`) to RTF, numerous MS Word formats (including MS Office Open XML), DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU Texinfo, MediaWiki, RTF, Groff, and other formats.
2. Doconce is a working strategy for never duplicating information. Text is written in a single place and then transformed to a number of different destinations of diverse type (software source code, manuals, tutorials, books, wikis, memos, emails, etc.). The Doconce markup language support this working strategy. The slogan is: "Write once, include anywhere".

Here are some Doconce features:

- Doconce markup does include tags, so the format is more tagged than Markdown and Pandoc, but less than reST, and very much less than LaTeX and HTML.
- Doconce can be converted to plain *untagged* text, often desirable for computer programs and email.
- Doconce has good support for copying in parts of computer code directly from the source code files via regular expressions for the start and end lines.
- Doconce has full support for LaTeX math and integrates well with big LaTeX projects (books).
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
- Contrary to the similar (and superior) Pandoc translator, Doconce supports Sphinx, Google wiki, Creole wiki (for bitbucket.org), lots of computer code environments in LaTeX, and a special exercise syntax. Doconce also also runs preprocessors (including Mako) such that the author can mix ordinary text with programming construction for generating parts of the text.

Doconce was particularly written for the following sample applications:

- Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.
- Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at web sites, and as LaTeX integrated in, e.g., a thesis.

- Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as Sphinx web pages, MS Word documents, or in wikis.

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover, Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

Disclaimer: Doconce is a simple tool, largely based on interpreting and handling text through regular expressions. The possibility for tweaking the layout is obviously limited since the text can go to all sorts of sophisticated markup languages. Moreover, because of limitations of regular expressions, some formatting of Doconce syntax may face problems when transformed to HTML, LaTeX, Sphinx, and similar formats.

CHAPTER

THREE

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 formating. Here are som examples.

- Bullet lists arise from lines starting with an asterisk.
- *Emphasized words* are surrounded by asterisks.
- **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 recognied 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 special support for advanced exercises features.
- With a simple 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 the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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 an 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 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 supperted, 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:

3.1 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 supperted, e.g.,

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

3.2 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

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

The pipe symbol acts as a delimiter between LaTeX code and the plain text version of the formula.

Blocks of mathematics are better typeset with raw LaTeX, inside `!bt` and `!et` (begin tex / end tex) instructions. The result looks like this:

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla^2 u + f, \\ \frac{\partial v}{\partial t} &= \nabla \cdot (q(u)\nabla v) + g \end{aligned} \tag{3.1}$$

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other 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. Such blocks look like

```
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.

It is possible to add a specification of an environment for typesetting the verbatim code block, e.g., `!bc xxx` where `xxx` is an identifier like `pycod` for code snippet in Python, `sys` for terminal session, etc. When Doconce is filtered to LaTeX, these identifiers are used as in `ptex2tex` and defined in a configuration file `.ptext2tex.cfg`, while when filtering to Sphinx, one can have a comment line in the Doconce file for mapping the identifiers to legal language names for Sphinx (which equals the legal language names for Pygments):

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

By default, `pro` and `cod` are `python`, `sys` is `console`, while `xpro` and `xcod` are computer language specific for `x` in `f` (Fortran), `c` (C), `cpp` (C++), `pl` (Perl), `m` (Matlab), `sh` (Unix shells), `cy` (Cython), and `py` (Python).

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!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. 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 the right writing enviroment for plain text.

3.3 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).

CHAPTER

FOUR

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
```

The `mako` or `preprocess` programs are always used to preprocess the file first, and options to `mako` or `preprocess` 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`. 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"`.

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.

4.1 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.

4.2 Pandoc

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 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. See the Pandoc documentation for the many features of the `pandoc` program.

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 `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> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

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

Quite some `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 makeing HTML notes with (some) mathematics.

4.3 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.

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`.

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 cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are about 40 styles to choose from.

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 -DLATEX_HEADING=traditional -DMINTED \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted 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}`). 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` or `fpro` are defined, 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. Here are some examples:

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
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 encourged to make a script for generating PDF from the LaTeX file.

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 use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code (specified, e.g., in the `*pro` and `*cod` environments in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the `minted` LaTeX package is needed. This package is included by running `doconce format` 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
```

4.4 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
```

4.5 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
```

4.6 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

4.7 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. You should at least read the script prior to executing it to have some idea of what is done.

The `doconce sphinx_dir` script copies directories named `figs` or `figures` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. If figures or movies are located in other directories, `automake-sphinx.py` must be edited accordingly. Files, to which there are local links (not `http:` or `file:` URLs), must be placed in the `_static` subdirectory of the Sphinx directory. The utility `doconce sphinxfix_localURLs` is run to check for local links in the Doconce file: for each such link, say `dir1/dir2/myfile.txt` it replaces the link by `_static/myfile.txt` and copies `dir1/dir2/myfile.txt` to a local `_static` directory (in the same directory as the script is run). However, we recommend instead that the writer of the document places files in `_static` or lets a script do it automatically. The user must copy all `_static/*` files to the `_static` subdirectory of the Sphinx directory. It may be wise to always put files, to which there are local links in the Doconce document, in a `_static` or `_static-name` directory and use these local links. Then links do not need to be modified when creating a Sphinx version of the document.

Doconce comes with a collection of HTML themes for Sphinx documents. These are packed out in the Sphinx directory, the `conf.py` configuration file for Sphinx is edited accordingly, and a script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics` and `pyramid`, one writes

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

The resulting directories with HTML documents are _build/html_fenics and _build/html_pyramid, respectively. Without arguments, make-themes.sh makes all available themes (!).

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.

4.8 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 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).

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. At the same time, the book can also be published as a standard LaTeX book or a Sphinx web document.

4.9 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.

4.10 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.

4.11 Dependencies and Installation

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://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..
```

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.

To make LaTeX documents (without going through the reStructuredText format) you need ptex2tex, which is installed by

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
```

As seen, `cp2texmf.sh` 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-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).

The *minted* LaTeX style is offered by `ptex2tex` and popular among users. This style requires the package Pygments:

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

If you use the minted style, you have to enable it by running `ptex2tex -DMINTED` and then `latex -shell-escape`, see the the section *From Doconce to Other Formats*.

For `rst` output and further transformation to LaTeX, HTML, XML, OpenOffice, and so on, one needs docutils. The installation 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 `sudo python setup.py install`.

Output to `sphinx` requires of course Sphinx, installed by

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

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 ..
```

Finally, translation to `pandoc` requires the Pandoc program (written in Haskell) to be installed.

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

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 itself is frequently updated so these commands should be run regularly.

CHAPTER

FIVE

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><section ids="doconce-document-once
-include-anywhere" names="doconce:\ document\ once,\ include\ anywhere"><title>D
oconce: Document Once, Include Anywhere</title><field_list><field><field_name>Au
thor</field_name><field_body><paragraph>Hans Petter Langtangen</paragraph></fiel
d_body></field><field><field_name>Date</field_name><field_body><paragraph>Jul 25
, 2012</paragraph><bullet_list bullet="*"><list_item><paragraph>When writing a n
ote, 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 dupnames="sphinx" ids="sphi
nx" refuri="http://sphinx.pocoo.org/contents.html"/>, and <reference name="wiki"
 refuri="http://code.google.com/p/support/wiki/WikiSyntax">wiki</reference><targ
et ids="wiki" names="wiki" refuri="http://code.google.com/p/support/wiki/WikiSyn
tax"/>? 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="the-doconce-concept" names="the
\ doconce\ concept"><title>The Doconce Concept</title><paragraph>Doconce is two
things:</paragraph><block_quote><enumerated_list enumtype="arabic" prefix="" suf
fix="."><list_item><paragraph>Doconce is a very simple and minimally tagged mark
up language that
looks like ordinary ASCII text (much like what you would use in an
email), but the text can be transformed to numerous other formats,
including HTML, Pandoc, Google wiki, LaTeX, PDF, reStructuredText
(reST), Sphinx, Epytext, and also plain text (where non-obvious
formatting/tags are removed for clear reading in, e.g.,
emails). From reST you can (via <literal>rst2*</literal> programs) go to XML, HT
ML,
LaTeX, PDF, OpenOffice, and from the latter (via <literal>unoconv</literal>) to
RTF, numerous MS Word formats (including MS Office Open XML),
DocBook, PDF, MediaWiki, XHTML. From Pandoc one can generate
Markdown, reST, LaTeX, HTML, PDF, DocBook XML, OpenOffice, GNU
Texinfo, MediaWiki, RTF, Groff, and other formats.</paragraph></list_item><list_
```

tutorial.xml

item><paragraph>Doconce is a working strategy for never duplicating information.
Text is written in a single place and then transformed to
a number of different destinations of diverse type (software
source code, manuals, tutorials, books, wikis, memos, emails, etc.).
The Doconce markup language support this working strategy.
The slogan is: "Write once, include anywhere".</paragraph></list_item>
</enumerated_list></block_quote><paragraph>Here are some Doconce features:</para
graph><block_quote><bullet_list bullet="*"><list_item><paragraph>Doconce markup
does include tags, so the format is more tagged than
Markdown and Pandoc, but less than reST, and very much less than
LaTeX and HTML.</paragraph></list_item><list_item><paragraph>Doconce can be conv
erted to plain <emphasis>untagged</emphasis> text,
often desirable for computer programs and email.</paragraph></list_item><list_it
em><paragraph>Doconce has good support for copying in parts of computer code
directly from the source code files via regular expressions
for the start and end lines.</paragraph></list_item><list_item><paragraph>Doconc
e has full support for LaTeX math and integrates well
with big LaTeX projects (books).</paragraph></list_item><list_item><paragraph>Do
conce is almost self-explanatory and is a handy starting point
for generating documents in more complicated markup languages, such
as Google wiki, LaTeX, and Sphinx. A primary application of Doconce
is just to make the initial versions of a Sphinx or wiki document.</paragraph></
list_item><list_item><paragraph>Contrary to the similar (and superior) Pandoc tr
anslator, Doconce
supports Sphinx, Google wiki, Creole wiki (for bitbucket.org),
lots of computer code environments in LaTeX, and a special exercise
syntax. Doconce also also runs preprocessors (including Mako)
such that the author can mix ordinary text with programming
construction for generating parts of the text.</paragraph></list_item></bullet_l
ist></block_quote><paragraph>Doconce was particularly written for the following
sample applications:</paragraph><block_quote><bullet_list bullet="*"><list_item>
<paragraph>Large books written in LaTeX, but where many pieces (computer demos,
projects, examples) can be written in Doconce to appear in other
contexts in other formats, including plain HTML, Sphinx, wiki, or MS Word.</para
graph></list_item><list_item><paragraph>Software documentation, primarily Python
 doc strings, which one wants
to appear as plain untagged text for viewing in Pydoc, as reStructuredText
for use with Sphinx, as wiki text when publishing the software at
web sites, and as LaTeX integrated in, e.g., a thesis.</paragraph></list_item><l
ist_item><paragraph>Quick memos, which start as plain text in email, then some s
mall
amount of Doconce tagging is added, before the memos can appear as
Sphinx web pages, MS Word documents, or in wikis.</paragraph></list_item></bulle
t_list></block_quote><paragraph>History: Doconce was developed in 2006 at a time
 when most popular
markup languages used quite some tagging. Later, almost untagged
markup languages like Markdown and Pandoc became popular. Doconce is
not a replacement of Pandoc, which is a considerably more
sophisticated project. Moreover, Doconce was developed mainly to
fulfill the needs for a flexible source code base for books with much
mathematics and computer code.</paragraph><paragraph>Disclaimer: Doconce is a si
mple tool, largely based on interpreting
and handling text through regular expressions. The possibility for
tweaking the layout is obviously limited since the text can go to
all sorts of sophisticated markup languages. Moreover, because of
limitations of regular expressions, some formatting of Doconce syntax
may face problems when transformed to HTML, LaTeX, Sphinx, and similar
formats.</paragraph></section><section ids="what-does-doconce-look-like" names="
what\ does\ doconce\ look\ like?"><title>What Does Doconce Look Like?</title><pa

tutorial.xml

```
ragraph>Doconce text looks like ordinary text, but there are some almost invisib
le
text constructions that allow you to control the formating. Here are
som examples.</paragraph><block_quote><bullet_list bullet="*"><list_item><paragr
aph>Bullet lists arise from lines starting with an asterisk.</paragraph></list_i
tem><list_item><paragraph><emphasis>Emphasized words</emphasis> are surrounded b
y asterisks.</paragraph></list_item><list_item><paragraph><strong>Words in boldf
ace</strong> are surrounded by underscores.</paragraph></list_item><list_item><p
aragraph>Words from computer code are enclosed in back quotes and
then typeset <literal>verbatim (in a monospace font)</literal>.</paragraph></lis
t_item><list_item><paragraph>Section headings are recognied by equality (<litera
l>=</literal>) signs before
and after the title, and the number of <literal>=</literal> signs indicates the
level of the section: 7 for main section, 5 for subsection, and
3 for subsubsection.</paragraph></list_item><list_item><paragraph>Paragraph head
ings are recognized by a double underscore
before and after the heading.</paragraph></list_item><list_item><paragraph>The a
bstract of a document starts with <emphasis>Abstract</emphasis> as paragraph
heading, and all text up to the next heading makes up the abstract,</paragraph><
/list_item><list_item><paragraph>Blocks of computer code can easily be included
by placing
<literal>!bc</literal> (begin code) and <literal>!ec</literal> (end code) comman
ds at separate lines
before and after the code block.</paragraph></list_item><list_item><paragraph>Bl
ocks of computer code can also be imported from source files.</paragraph></list_
item><list_item><paragraph>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.</paragraph></list_item><list_item><paragraph>Th
ere is support for both LaTeX and text-like inline mathematics.</paragraph></lis
t_item><list_item><paragraph>Figures and movies with captions, simple tables,
URLs with links, index list, labels and references are supported.</paragraph></l
ist_item><list_item><paragraph>Invisible comments in the output format can be in
serted throughout
the text.</paragraph></list_item><list_item><paragraph>Visible comments can be i
nserted so that authors and readers can
comment upon the text (and at any time turn on/off output of such
comments).</paragraph></list_item><list_item><paragraph>There is special support
 for advanced exercises features.</paragraph></list_item><list_item><paragraph>W
ith a simple preprocessor, Preprocess or Mako, one can include
other documents (files) and large portions of text can be defined
in or out of the text.</paragraph></list_item><list_item><paragraph>With the Mak
o preprocessor one can even embed Python
code and use this to steer generation of Doconce text.</paragraph></list_item></
bullet_list></block_quote><paragraph>Here is an example of some simple text writ
ten in the Doconce format:</paragraph><literal_block xml:space="preserve">=====
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 automatically numbered items instead of bullets,
```

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

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

References to sections may use logical names as labels (e.g., a
&quot;label&quot; 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 supperted, 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</literal_block><paragraph>The Doconce
 text above results in the following little document:</paragraph><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 <lit
eral>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>UR
Ls with a link word are possible, as in <reference name="hpl" refuri="http://fol
k.uio.no/hpl">hpl</reference><target ids="hpl" names="hpl" refuri="http://folk.u
io.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="tutori
al.do.txt"/>.</paragraph><paragraph>References to sections may use logical names
 as labels (e.g., a
&quot;label&quot; 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><paragra
ph>Doconce also allows inline comments such as (<strong>hpl</strong>: here I wil
```

tutorial.xml

```
l 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 supperted, e.g.,</paragraph><table><tgro
up cols="3"><colspec colwidth="12"/><colspec colwidth="12"/><colspec colwidth="1
2"/><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><
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><paragr
aph>14.717624</paragraph></entry></row></tbody></tgroup></table></section><secti
on 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.</paragraph><paragraph>Blocks of mathematics are better t
ypeset with raw LaTeX, inside
<literal>!bt</literal> and <literal>!et</literal> (begin tex / end tex) instruct
ions.
The result looks like this:</paragraph><literal_block xml:space="preserve">\begi
n{eqnarray}
{\partial u\over\partial t} &amp;=&amp; \nabla^2 u + f, label{myeq1}\\
{\partial v\over\partial t} &amp;=&amp; \nabla\cdot(q(u)\nabla v) + g
\end{eqnarray}</literal_block><paragraph>Of course, such blocks only looks nice
in LaTeX. The raw
LaTeX syntax appears in all other formats (but can still be useful
for those who can read LaTeX syntax).</paragraph><paragraph>You can have blocks
of computer code, starting and ending with
<literal>!bc</literal> and <literal>!ec</literal> instructions, respectively. Su
ch blocks look like:</paragraph><literal_block xml:space="preserve">from math im
port sin, pi
def myfunc(x):
    return sin(pi*x)

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
>It is possible to add a specification of an
environment for typesetting the verbatim code block, e.g., <literal>!bc xxx</lit
eral>
where <literal>xxx</literal> is an identifier like <literal>pycod</literal> for
code snippet in Python,
<literal>sys</literal> for terminal session, etc. When Doconce is filtered to La
TeX,
these identifiers are used as in <literal>ptex2tex</literal> and defined in a
configuration file <literal>.ptext2tex.cfg</literal>, while when filtering
```

tutorial.xml

```
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):</paragraph><literal_block xml:space="pre
serve"># sphinx code-blocks: pycod=python cod=fortran cppcod=c++ sys=console</li
teral_block><paragraph>By default, <literal>pro</literal> and <literal>cod</lite
ral> are <literal>python</literal>, <literal>sys</literal> is <literal>console</
literal>,
while <literal>xpro</literal> and <literal>xcod</literal> are computer language
specific for <literal>x</literal>
in <literal>f</literal> (Fortran), <literal>c</literal> (C), <literal>cpp</liter
al> (C++), <literal>pl</literal> (Perl), <literal>m</literal> (Matlab),
<literal>sh</literal> (Unix shells), <literal>cy</literal> (Cython), and <litera
l>py</literal> (Python).</paragraph><comment xml:space="preserve">(Any sphinx co
de-block comment, whether inside verbatim code</comment><comment xml:space="pres
erve">blocks or outside, yields a mapping between bc arguments</comment><comment
 xml:space="preserve">and computer languages. In case of muliple definitions, th
e</comment><comment xml:space="preserve">first one is used.)</comment><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!). A complete file is typeset
with <literal>!bc pro</literal>, while a part of a file is copied into a <litera
l>!bc cod</literal>
environment. What <literal>pro</literal> and <literal>cod</literal> mean is then
 defined through
a <literal>.ptex2tex.cfg</literal> file for LaTeX and a <literal>sphinx code-blo
cks</literal>
comment for Sphinx.</paragraph><paragraph>Another document can be included by wr
iting <literal>#include &amp;quot;mynote.do.txt&amp;quot;</literal>
on a line starting with (another) hash sign.  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 the
right writing enviroment for plain text.</paragraph><target refid="newcommands"/
></section><section ids="macros-newcommands-cross-references-index-and-bibliogra
phy newcommands" names="macros\ (newcommands),\ cross-references,\ index,\ and\
bibliography newcommands"><title>Macros (Newcommands), Cross-References, Index,
and Bibliography</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
```

tutorial.xml

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
efid="doconce2formats"/></section></section><section ids="from−doconce−to−other−
formats doconce2formats" names="from\ doconce\ to\ other\ formats doconce2format
s"><title>From Doconce to Other Formats</title><paragraph>Transformation of a Do
conce document <literal>mydoc.do.txt</literal> to various other
formats applies the script <literal>doconce format</literal>:</paragraph><litera
l_block xml:space="preserve">Terminal> doconce format format mydoc.do.txt</li
teral_block><paragraph>or just:</paragraph><literal_block xml:space="preserve">T
erminal> doconce format format mydoc</literal_block><paragraph>The <literal>m
ako</literal> or <literal>preprocess</literal> programs are always used to prepr
ocess the
file first, and options to <literal>mako</literal> or <literal>preprocess</liter
al> can be added after the
filename. For example:</paragraph><literal_block xml:space="preserve">Terminal&g
t; doconce format latex mydoc −Dextra_sections −DVAR1=5   # preprocess
Terminal> doconce format latex yourdoc extra_sections=True VAR1=5 # mako</li
teral_block><paragraph>The variable <literal>FORMAT</literal> is always defined
as the current format when
running <literal>preprocess</literal>. That is, in the last example, <literal>FO
RMAT</literal> is
defined as <literal>latex</literal>. Inside the Doconce document one can then pe
rform
format specific actions through tests like <literal>#if FORMAT == "latex&qu
ot;</literal>.</paragraph><paragraph>Inline comments in the text are removed fro
m the output by:</paragraph><literal_block xml:space="preserve">Terminal> doc
once 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> doc
once remove_inline_comments mydoc</literal_block><paragraph>This action is conve
nient when a Doconce document reaches its final form
and comments by different authors should be removed.</paragraph><section dupname
s="html" ids="id1"><title>HTML</title><paragraph>Making an HTML version of a Doc
once file <literal>mydoc.do.txt</literal>
is performed by:</paragraph><literal_block xml:space="preserve">Terminal> doc
once format html mydoc</literal_block><paragraph>The resulting file <literal>myd
oc.html</literal> can be loaded into any web browser for viewing.</paragraph></s
ection><section dupnames="pandoc" ids="pandoc"><title>Pandoc</title><paragraph>O
utput in Pandoc’s extended Markdown format results from:</paragraph><literal_blo
ck 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 mydoc.mkd
</literal_block><paragraph>Pandoc supports <literal>latex</literal>, <literal>ht

tutorial.xml

```
ml</literal>, <literal>odt</literal> (OpenOffice), <literal>docx</literal> (Micr
osoft
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.
See the <reference name="Pandoc documentation" refuri="http://johnmacfarlane.net
/pandoc/README.html">Pandoc documentation</reference><target ids="pandoc-documen
tation" names="pandoc\ documentation" refuri="http://johnmacfarlane.net/pandoc/R
EADME.html"/>
for the many features of the <literal>pandoc</literal> program.</paragraph><para
graph>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>pan
doc</literal>, or
<literal>doconce format latex</literal>, and then going from LaTeX to the desire
d format
using <literal>pandoc</literal>.
Here is an example on the latter strategy:</paragraph><literal_block xml:space="
preserve">Terminal&gt; doconce format latex mydoc
Terminal&gt; doconce ptex2tex mydoc
Terminal&gt; pandoc -f latex -t docx -o mydoc.docx mydoc.tex</literal_block><par
agraph>When we go through <literal>pandoc</literal>, only single equations or <l
iteral>align*</literal>
environments are well understood.</paragraph><paragraph>Quite some <literal>doco
nce 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.</paragraph><paragraph>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:</paragraph><literal_block xml:space=
"preserve">Terminal&gt; doconce format pandoc mydoc
Terminal&gt; pandoc -t html -o mydoc.html -s --mathjax mydoc.mkd</literal_block>
<paragraph>The <literal>-s</literal> option adds a proper header and footer to t
he <literal>mydoc.html</literal> file.
This recipe is a quick way of makeing HTML notes with (some) mathematics.</parag
raph></section><section dupnames="latex" ids="id2"><title>LaTeX</title><paragrap
h>Making a LaTeX file <literal>mydoc.tex</literal> from <literal>mydoc.do.txt</l
iteral> is done in two steps:
.. Note: putting code blocks inside a list is not successful in many</paragraph>
<comment xml:space="preserve">formats - the text may be messed up. A better choi
ce is a paragraph</comment><comment xml:space="preserve">environment, as used he
re.</comment><paragraph><emphasis>Step 1.</emphasis> 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>)
:</paragraph><literal_block xml:space="preserve">Terminal&gt; doconce format lat
ex mydoc</literal_block><paragraph>LaTeX-specific commands (&amp;quot;newcommands&qu
ot;) 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 <reference name="Mac
ros (Newcommands), Cross-References, Index, and Bibliography" refid="macros-newc
ommands-cross-references-index-and-bibliography">Macros (Newcommands), Cross-Ref
erences, Index, and Bibliography</reference>).
If these files are present, they are included in the LaTeX document
so that your commands are defined.</paragraph><paragraph><emphasis>Step 2.</emph
asis> Run <literal>ptex2tex</literal> (if you have it) to make a standard LaTeX
file:</paragraph><literal_block xml:space="preserve">Terminal&gt; ptex2tex mydoc
```

tutorial.xml

```
</literal_block><paragraph>In case you do not have <literal>ptex2tex</literal>,
you may run a (very) simplified version:</paragraph><literal_block xml:space="pr
eserve">Terminal&gt; doconce ptex2tex mydoc</literal_block><paragraph>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:</paragraph><literal_block xml:space="preserve">Termin
al&gt; ptex2tex -DHELVETICA mydoc
Terminal&gt; doconce ptex2tex mydoc -DHELVETICA  # alternative</literal_block><p
aragraph>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 &amp;quot;maketitle&amp;quot; headin
g
is also available through <literal>-DLATEX_HEADING=traditional</literal>.
A separate titlepage can be generate by
<literal>-DLATEX_HEADING=titlepage</literal>.</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 ca
n
insert verbatim block styles as defined in your <literal>.ptex2tex.cfg</literal>
file, e.g., <literal>!bc cod</literal> for a code snippet, where <literal>cod</l
iteral> is set to
a certain environment in <literal>.ptex2tex.cfg</literal> (e.g., <literal>CodeIn
tended</literal>).
There are about 40 styles to choose from.</paragraph><paragraph>Also the <litera
l>doconce ptex2tex</literal> command supports preprocessor directives
for processing the <literal>.p.tex</literal> file. The command allows specificat
ions
of code environments as well. Here is an example:</paragraph><literal_block xml:
space="preserve">Terminal&gt; doconce ptex2tex -DLATEX_HEADING=traditional -DMIN
TED \
          &amp;quot;sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}&amp;quot
; \
          fpro=minted fcod=minted envir=ans:nt</literal_block><paragraph>Note th
at <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>\end{minted}</literal>).
Specifying <literal>envir=ans:nt</literal> means that all other environments are
 typeset
with 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> or <literal>fpro</literal> are
defined, the plain <literal>\begin{verbatim}</literal> and <literal>\end{verbati
m}</literal> used.</paragraph><paragraph><emphasis>Step 2b (optional).</emphasis
> Edit the <literal>mydoc.tex</literal> file to your needs.
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.
Here are some examples:</paragraph><literal_block xml:space="preserve">Terminal&
gt; doconce replace 'section{' 'section*{' mydoc.tex
Terminal&gt; doconce subst 'title\{(.+)Using (.+)\}' \
```

tutorial.xml


```
          'title{\g&lt;1&gt; \\\\ [1.5mm] Using \g&lt;2&gt;' 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 encourged to make a script for
generating PDF from the LaTeX file.</paragraph><paragraph><emphasis>Step 3.</emp
hasis> 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 use the <li
teral>Minted_Python</literal>, <literal>Minted_Cpp</literal>, etc.,
environments in <literal>ptex2tex</literal> for typesetting code (specified, e.g
., in
the <literal>*pro</literal> and <literal>*cod</literal> environments in <literal
>.ptex2tex.cfg</literal> or
<literal>$HOME/.ptex2tex.cfg</literal>), the <literal>minted</literal> LaTeX pac
kage is needed.  This
package is included by running <literal>doconce format</literal> with the <liter
al>-DMINTED</literal>
option:</paragraph><literal_block xml:space="preserve">Terminal&gt; ptex2tex -DM
INTED mydoc</literal_block><paragraph>In this case, <literal>latex</literal> mus
t be run with the
<literal>-shell-escape</literal> option:</paragraph><literal_block xml:space="pr
eserve">Terminal&gt; latex -shell-escape mydoc
Terminal&gt; latex -shell-escape mydoc
Terminal&gt; makeindex mydoc   # if index
Terminal&gt; bibitem mydoc     # if bibliography
Terminal&gt; latex -shell-escape mydoc
Terminal&gt; dvipdf mydoc</literal_block></section><section ids="pdflatex" names
="pdflatex"><title>PDFLaTeX</title><paragraph>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&gt; 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&gt; pdflatex -shell-escape mydoc
Terminal&gt; makeindex mydoc   # if index
Terminal&gt; bibitem mydoc     # if bibliography
Terminal&gt; pdflatex -shell-escape mydoc</literal_block></section><section ids=
"plain-ascii-text" names="plain\ ascii\ text"><title>Plain ASCII Text</title><pa
ragraph>We can go from Doconce &amp;quot;back to&amp;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&gt
; doconce format plain mydoc.do.txt  # results in mydoc.txt</literal_block></sec
tion><section dupnames="restructuredtext" ids="id3"><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&gt; doconce format rst mydoc.do.txt</literal_block><
paragraph>We may now produce various other formats:</paragraph><literal_block xm
l:space="preserve">Terminal&gt; rst2html.py  mydoc.rst &gt; mydoc.html # html
Terminal&gt; rst2latex.py mydoc.rst &gt; mydoc.tex  # latex
Terminal&gt; rst2xml.py   mydoc.rst &gt; mydoc.xml  # XML
Terminal&gt; rst2odt.py   mydoc.rst &gt; mydoc.odt  # OpenOffice</literal_block>
<paragraph>The OpenOffice file <literal>mydoc.odt</literal> can be loaded into O
```

tutorial.xml

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
"MS Word world". 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
rums.org/showthread.php?t=1033441" refuri="http://ubuntuforums.org/showthread.ph
p?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-blogspot-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="id4"><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="authors' names" \
          title="some title" 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>

tutorial.xml

```
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&gt; 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.
You should at least read the script prior to executing it to have
some idea of what is done.</paragraph><paragraph>The <literal>doconce sphinx_dir
</literal> script copies directories named <literal>figs</literal> or
<literal>figures</literal> over to the Sphinx directory so that figures are acce
ssible
in the Sphinx compilation.  If figures or movies are located in other
directories, <literal>automake-sphinx.py</literal> must be edited accordingly.
Files,
to which there are local links (not <literal>http:</literal> or <literal>file:</
literal> URLs), must be
placed in the <literal>_static</literal> subdirectory of the Sphinx directory. T
he
utility <literal>doconce sphinxfix_localURLs</literal> is run to check for local
 links
in the Doconce file: for each such link, say <literal>dir1/dir2/myfile.txt</lite
ral> it
replaces the link by <literal>_static/myfile.txt</literal> and copies
<literal>dir1/dir2/myfile.txt</literal> to a local <literal>_static</literal> di
rectory (in the same
directory as the script is run).  However, we recommend instead that
the writer of the document places files in <literal>_static</literal> or lets a
script
do it automatically. The user must copy all <literal>_static/*</literal> files t
o the
<literal>_static</literal> subdirectory of the Sphinx directory.  It may be wise
 to
always put files, to which there are local links in the Doconce
document, in a <literal>_static</literal> or <literal>_static-name</literal> dir
ectory and use these
local links. Then links do not need to be modified when creating a
Sphinx version of the document.</paragraph><paragraph>Doconce comes with a colle
ction of HTML themes for Sphinx documents.
These are packed out in the Sphinx directory, the <literal>conf.py</literal>
configuration file for Sphinx is edited accordingly, and a script
<literal>make-themes.sh</literal> can make HTML documents with one or more theme
s.
For example,
to realize the themes <literal>fenics</literal> and <literal>pyramid</literal>,
one writes:</paragraph><literal_block xml:space="preserve">Terminal&gt; ./make-t
hemes.sh fenics pyramid</literal_block><paragraph>The resulting directories with
 HTML documents are <literal>_build/html_fenics</literal>
and <literal>_build/html_pyramid</literal>, respectively. Without arguments,
<literal>make-themes.sh</literal> makes all available themes (!).</paragraph><pa
ragraph>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 <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&gt; 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;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&gt; 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
```

```
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.</paragraph><paragraph><emphasis>Step 6.</emphasis> View the r
esult:</paragraph><literal_block xml:space="preserve">Terminal&gt; firefox _buil
d/html/index.html</literal_block><paragraph>Note that verbatim code blocks can b
e typeset in a variety of ways
depending the argument that follows <literal>!bc</literal>: <literal>cod</litera
l> gives Python
(<literal>code-block:: python</literal> in Sphinx syntax) and <literal>cppcod</l
iteral> gives C++, but
all such arguments can be customized both for Sphinx and LaTeX output.</paragrap
h></section><section ids="wiki-formats" names="wiki\ formats"><title>Wiki Format
s</title><paragraph>There are many different wiki formats, but Doconce only supp
orts three:
<reference name="Googlecode wiki" refuri="http://code.google.com/p/support/wiki/
WikiSyntax">Googlecode wiki</reference><target ids="googlecode-wiki" names="goog
lecode\ wiki" refuri="http://code.google.com/p/support/wiki/WikiSyntax"/>, Media
Wiki, and Creole Wiki. These formats are called
<literal>gwiki</literal>, <literal>mwiki</literal>, and <literal>cwiki</literal>
, respectively.
Transformation from Doconce to these formats is done by:</paragraph><literal_blo
ck xml:space="preserve">Terminal&gt; doconce format gwiki mydoc.do.txt
Terminal&gt; doconce format mwiki mydoc.do.txt
Terminal&gt; doconce format cwiki mydoc.do.txt</literal_block><paragraph>The Goo
glecode wiki document, <literal>mydoc.gwiki</literal>, is most conveniently stor
ed
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.</paragraph><paragraph>When the Doconce file contai
ns 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).</paragraph><par
agraph>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.wikibo
oks.org">Wikibooks</reference><target ids="wikibooks" names="wikibooks" refuri="
http://en.wikibooks.org"/>
and publish these in PDF and MediaWiki format.
At the same time, the book can also be published as a
standard LaTeX book or a Sphinx web document.</paragraph></section><section ids=
"tweaking-the-doconce-output" names="tweaking\ the\ doconce\ output"><title>Twea
king the Doconce Output</title><paragraph>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 im
age 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
```

tutorial.xml

```
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.</paragraph></
section><section ids="demos" names="demos"><title>Demos</title><paragraph>The cu
rrent text is generated from a Doconce format stored in the file:</paragraph><li
teral_block xml:space="preserve">docs/tutorial/tutorial.do.txt</literal_block><p
aragraph>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 gen
erated
files and comparing them with the original <literal>tutorial.do.txt</literal> fi
le,
gives a quick introduction to how Doconce is used in a real case.
<reference name="Here" refuri="https://doconce.googlecode.com/hg/doc/demos/tutor
ial/index.html">Here</reference><target ids="here" names="here" refuri="https://
doconce.googlecode.com/hg/doc/demos/tutorial/index.html"/>
is a sample of how this tutorial looks in different formats.</paragraph><paragra
ph>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 translation
s.</paragraph></section><section ids="dependencies-and-installation" names="depe
ndencies\ and\ installation"><title>Dependencies and Installation</title><paragr
aph>Doconce itself is pure Python code hosted at <reference name="http://code.go
ogle.com/p/doconce" refuri="http://code.google.com/p/doconce">http://code.google
.com/p/doconce</reference><target ids="http-code-google-com-p-doconce" names="ht
tp://code.google.com/p/doconce" refuri="http://code.google.com/p/doconce"/>.  It
s installation from the
Mercurial (<literal>hg</literal>) source follows the standard procedure:</paragr
aph><literal_block xml:space="preserve"># Doconce
hg clone https://doconce.googlecode.com/hg/ doconce
cd doconce
sudo python setup.py install
cd ..</literal_block><paragraph>If you make use of the <reference name="Preproce
ss" refuri="http://code.google.com/p/preprocess">Preprocess</reference><target i
ds="preprocess" names="preprocess" refuri="http://code.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><targe
t ids="source-code" names="source\ code" refuri="http://pypi.python.org/pypi/pip
"/>.</paragraph><paragraph>To make LaTeX
documents (without going through the reStructuredText format) you
```

tutorial.xml

```
need <reference name="ptex2tex" refuri="http://code.google.com/p/ptex2tex">ptex2
tex</reference><target ids="ptex2tex" names="ptex2tex" refuri="http://code.googl
e.com/p/ptex2tex"/>, which is
installed by:</paragraph><literal_block xml:space="preserve">svn checkout http:/
/ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..</literal_block><paragraph>As seen, <literal>cp2texmf.sh</literal> copie
s some special stylefiles that
that <literal>ptex2tex</literal> potentially makes use of. Some more standard st
ylefiles
are also needed. These are installed by:</paragraph><literal_block xml:space="pr
eserve">sudo apt-get install texlive-latex-extra</literal_block><paragraph>on De
bian 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>The <emphasis>minted</emphasis> LaTeX style is offered by <literal>p
tex2tex</literal> and popular among
users. This style requires the package <reference name="Pygments" refuri="http:/
/pygments.org">Pygments</reference><target ids="pygments" names="pygments" refur
i="http://pygments.org"/>:</paragraph><literal_block xml:space="preserve">hg clo
ne ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments
cd pygments
sudo python setup.py install</literal_block><paragraph>If you use the minted sty
le, you have to enable it by running
<literal>ptex2tex -DMINTED</literal> and then <literal>latex -shell-escape</lite
ral>, see
the the section <reference name="From Doconce to Other Formats" refid="from-doco
nce-to-other-formats">From Doconce to Other Formats</reference>.</paragraph><par
agraph>For <literal>rst</literal> output and further transformation to LaTeX, HT
ML, XML,
OpenOffice, and so on, one needs <reference name="docutils" refuri="http://docut
ils.sourceforge.net">docutils</reference><target ids="docutils" names="docutils"
 refuri="http://docutils.sourceforge.net"/>.
The installation can be done by:</paragraph><literal_block xml:space="preserve">
svn checkout http://docutils.svn.sourceforge.net/svnroot/docutils/trunk/docutils
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 <literal>sudo python setup.py install</literal>.</paragraph><system_message
 backrefs="id5" level="2" line="403" source="tutorial.rst" type="WARNING"><paragr
aph>Duplicate explicit target name: &amp;quot;sphinx&amp;quot;.</paragraph></system_mess
age><paragraph>Output to <literal>sphinx</literal> requires of course <reference
 name="Sphinx" refuri="http://sphinx.pocoo.org">Sphinx</reference><target dupnam
es="sphinx" ids="id5" refuri="http://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
```

tutorial.xml

```
cd ..</literal_block><paragraph>When the output format is <literal>epydoc</liter
al> one needs that 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>Finally, translation to <literal>pandoc</literal
> requires the
<reference name="Pandoc" refuri="http://johnmacfarlane.net/pandoc/">Pandoc</refe
rence><target ids="id6" names="pandoc" refuri="http://johnmacfarlane.net/pandoc/
"/> program
(written in Haskell) to be installed:</paragraph><literal_block xml:space="prese
rve">sudo apt-get install pandoc</literal_block><paragraph><emphasis>Remark.</em
phasis> Several of the packages 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
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>.
Doconce itself is frequently updated so these commands should be
run regularly.</paragraph></section></section></document>
```