tutorial.do.txt

```
TITLE: Doconce: Document Once, Include Anywhere
AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of O
slo
DATE: August 25, 2010


# lines beginning with # are comment lines

 * 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 at some later stage eventually go
   with a particular format?

 * Do you find it problematic that 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
   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:

  o Doconce is a working strategy for documenting software in a single
    place and avoiding duplication of information. The slogan is:
    "Write once, include anywhere". This requires that what you write
    can be transformed to many different formats for a variety of
    documents (manuals, tutorials, books, doc strings, source code
    documentation, etc.).

  o Doconce is a simple and minimally tagged markup language that can
    be used for the above purpose. That is, the Doconce format look
    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, 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 reStructuredText
    you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the
    latter to RTF and MS Word.


===== 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. For example,

  * 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,
```

```
  * blocks of computer code can easily be included, also from source files,

  * blocks of LaTeX mathematics can easily be included,

  * there is support for both LaTeX and text-like inline mathematics,

  * figures with captions, URLs with links, labels and references
    are supported,

  * comments can be inserted throughout the text,

  * a preprocessor (much like the C preprocessor) is integrated so
    other documents (files) can be included and large portions of text
    can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|
!ec
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:

  o item 1
  o item 2
  o item 3

URLs with a link word are possible, as in http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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 $\nu = \sin(x)$|$v = sin(x)$,
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
from math import sin, pi
```

tutorial.do.txt

```
def myfunc(x):
    return sin(pi*x)

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

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

Another document can be included by writing ‘#include "mynote.do.txt"‘
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) =====

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).



# Example on including another Doconce file:

# #include "_doconce2anything.do.txt"


=== Demos ===

The current text is generated from a Doconce format stored in the file
!bc
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.
```

tutorial.do.txt

```
https://doconce.googlecode.com/hg/trunk/docs/demo/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 ===

Doconce needs the Python packages
http://docutils.sourceforge.net<docutils>,
http://code.google.com/p/preprocess<preprocess>, and
http://code.google.com/p/ptex2tex<ptex2tex>. The latter is only
needed for the LaTeX formats.


===== The Doconce Documentation Strategy for User Manuals =====

Doconce was particularly made for writing tutorials or user manuals
associated with computer codes. The text is written in Doconce format
in separate files. LaTeX, HTML, XML, and other versions of the text
is easily produced by the ‘doconce2format‘ script and standard tools.
A plain text version is often wanted for the computer source code,
this is easy to make, and then one can use
‘#include‘ statements in the computer source code to automatically
get the manual or tutorial text in comments or doc strings.
Below is a worked example.

Consider an example involving a Python module in a ‘basename.p.py‘ file.
The ‘.p.py‘ extension identifies this as a file that has to be
preprocessed) by the ‘preprocess‘ program.
In a doc string in ‘basename.p.py‘ we do a preprocessor include
in a comment line, say
!bc
#    #include "docstrings/doc1.dst.txt
!ec
#
# Note: we insert an error right above as the right quote is missing.
# Then preprocess skips the statement, otherwise it gives an error
# message about a missing file docstrings/doc1.dst.txt (which we don’t
# have, it’s just a sample file name). Also note that comment lines
# must not come before a code block for the rst/st/epytext formats to work.
#
The file ‘docstrings/doc1.dst.txt‘ is a file filtered to a specific format
(typically plain text, reStructedText, or Epytext) from an original
"singleton" documentation file named ‘docstrings/doc1.do.txt‘. The ‘.dst.txt‘
is the extension of a file filtered ready for being included in a doc
string (‘d‘ for doc, ‘st‘ for string).

For making an Epydoc manual, the ‘docstrings/doc1.do.txt‘ file is
filtered to ‘docstrings/doc1.epytext‘ and renamed to
‘docstrings/doc1.dst.txt‘.  Then we run the preprocessor on the
‘basename.p.py‘ file and create a real Python file
‘basename.py‘. Finally, we run Epydoc on this file. Alternatively, and
nowadays preferably, we use Sphinx for API documentation and then the
Doconce ‘docstrings/doc1.do.txt‘ file is filtered to
‘docstrings/doc1.rst‘ and renamed to ‘docstrings/doc1.dst.txt‘. A
Sphinx directory must have been made with the right ‘index.rst‘ and
‘conf.py‘ files. Going to this directory and typing ‘make html‘ makes
```

tutorial.do.txt

```
the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For
this purpose we filter ‘docstrings/doc1.do.txt‘ to plain text format
(‘docstrings/doc1.txt‘) and rename to ‘docstrings/doc1.dst.txt‘. The
preprocessor transforms the ‘basename.p.py‘ file to a standard Python
file ‘basename.py‘. The doc strings are now in plain text and well
suited for Pydoc or reading by humans. All these steps are automated
by the ‘insertdocstr.py‘ script.  Here are the corresponding Unix
commands:

!bc
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
!ec


======= Warning/Disclaimer =======

Doconce can be viewed is a unified interface to a variety of
typesetting formats.  This interface is minimal in the sense that a
lot of typesetting features are not supported, for example, footnotes
and bibliography. For many documents the simple Doconce format is
sufficient, while in other cases you need more sophisticated
formats. Then you can just filter the Doconce text to a more
approprite format and continue working in this format only.  For
example, reStructuredText is a good alternative: it is more tagged
than Doconce and cannot be filtered to plain, untagged text, or wiki,
and the LaTeX output is not at all as clean, but it also has a lot
```

```
more typesetting and tagging features than Doconce.
```

Doconce: Document Once, Include Anywhere

Hans Petter Langtangen
Simula Research Laboratory and University of Oslo

August 25, 2010

- 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 at some later stage eventually go with a particular format?
- Do you find it problematic that 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 place, and include it anywhere?

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

0.1 The Doconce Concept

Doconce is two things:

1. Doconce is a working strategy for documenting software in a single place and avoiding duplication of information. The slogan is: "Write once, include anywhere". This requires that what you write can be transformed to many different formats for a variety of documents (manuals, tutorials, books, doc strings, source code documentation, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. That is, the Doconce format look 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, 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 reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.

0.2 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. For example,

- 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,

- blocks of computer code can easily be included, also from source files,
- blocks of LaTeX mathematics can easily be included,
- there is support for both LaTeX and text-like inline mathematics,
- figures with captions, URLs with links, labels and references are supported,
- comments can be inserted throughout the text,
- a preprocessor (much like the C preprocessor) is integrated so other documents (files) can be included and large portions of text can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|
```

The Doconce text above results in the following little document:

0.3 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. Just a file link goes like 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 Chapter **??**.

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

0.4 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)
```

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

Another document can be included by writing `#include "mynote.do.txt"` 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.

0.5 Macros (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).

1 From Doconce to Other Formats

Transformation of a Doconce document to various other formats applies the script `doconce2format`:

```
Unix/DOS> doconce2format format mydoc.do.txt
```

The `preprocess` program is always used to preprocess the file first, and options to `preprocess` can be added after the filename. For example,

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections
```

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"`.

1.1 HTML

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

```
Unix/DOS> doconce2format HTML mydoc.do.txt
```

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

1.2 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 `ptex2tex`:

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in a file `newcommands.tex`. If this file is present, it is 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,

```
Unix/DOS> ptex2tex mydoc
```

or just perform a plain copy,

```
Unix/DOS> cp mydoc.p.tex mydoc.tex
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. Finally, compile `mydoc.tex` the usual way and create the PDF file.

1.3 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:

```
Unix/DOS> doconce2format plain mydoc.do.txt  # results in mydoc.txt
```

1.4 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`:

```
Unix/DOS> doconce2format rst mydoc.do.txt
```

We may now produce various other formats:

```
Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
Unix/DOS> 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. That is, one can easily go from Doconce to Microsoft Word.

1.5 Sphinx

Sphinx documents can be created from a Doconce source in a few steps.

Step 1. Translate Doconce into the Sphinx dialect of the reStructuredText format:

```
Unix/DOS> doconce2format sphinx mydoc.do.txt
```

Step 2. Create a Sphinx root directory with a `conf.py` file, 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
```

Step 3. Move the `tutorial.rst` file to the Sphinx root directory:

```
Unix/DOS> mv mydoc.rst sphinx-rootdir
```

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
```

Many other formats are also possible.

Step 6. View the result:

```
Unix/DOS> firefox _build/html/index.html
```

1.6 Google Code Wiki

There are several different wiki dialects, but Doconce only support the one used by Google Code. The transformation to this format, called `gwiki` to explicitly mark it as the Google Code dialect, is done by

```
Unix/DOS> doconce2format gwiki mydoc.do.txt
```

You can then open a new wiki page for your Google Code project, copy the `mydoc.gwiki` output file from `doconce2format` and paste the file contents into the wiki page. Press **Preview** or **Save Page** to see the formatted result.

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

```
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. Doconce needs the Python packages docutils, preprocess, and ptex2tex. The latter is only needed for the LaTeX formats.

1.7 The Doconce Documentation Strategy for User Manuals

Doconce was particularly made for writing tutorials or user manuals associated with computer codes. The text is written in Doconce format in separate files. LaTeX, HTML, XML, and other versions of the text is easily produced by the `doconce2format` script and standard tools. A plain text version is often wanted for the computer source code, this is easy to make, and then one can use `#include` statements in the computer source code to automatically get the manual or tutorial text in comments or doc strings. Below is a worked example.

Consider an example involving a Python module in a `basename.p.py` file. The `.p.py` extension identifies this as a file that has to be preprocessed) by the `preprocess` program. In a doc string in `basename.p.py` we do a preprocessor include in a comment line, say

```
#    #include "docstrings/doc1.dst.txt
```

The file `docstrings/doc1.dst.txt` is a file filtered to a specific format (typically plain text, reStructedText, or Epytext) from an original "singleton" documentation file named `docstrings/doc1.do.txt`. The `.dst.txt` is the extension of a file filtered ready for being included in a doc string (`d` for doc, `st` for string).

For making an Epydoc manual, the `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.epytext` and renamed to `docstrings/doc1.dst.txt`. Then we run the preprocessor on the `basename.p.py` file and create a real Python file `basename.py`. Finally, we run Epydoc on this file. Alternatively, and nowadays preferably, we use Sphinx for API documentation and then the Doconce `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.rst` and renamed to `docstrings/doc1.dst.txt`. A Sphinx directory must have been made with the right `index.rst` and `conf.py` files. Going to this directory and typing `make html` makes the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For this purpose we filter `docstrings/doc1.do.txt` to plain text format (`docstrings/doc1.txt`) and rename to `docstrings/doc1.dst.txt`. The preprocessor transforms the `basename.p.py` file to a standard Python file `basename.py`. The doc strings are now in plain text and well suited for Pydoc or reading by humans. All these steps are automated by the `insertdocstr.py` script. Here are the corresponding Unix commands:

```
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
```

2 Warning/Disclaimer

Doconce can be viewed is a unified interface to a variety of typesetting formats. This interface is minimal in the sense that a lot of typesetting features are not supported, for example, footnotes and bibliography. For many documents the simple Doconce

format is sufficient, while in other cases you need more sophisticated formats. Then you can just filter the Doconce text to a more approprite format and continue working in this format only. For example, reStructuredText is a good alternative: it is more tagged than Doconce and cannot be filtered to plain, untagged text, or wiki, and the LaTeX output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

Doconce: Document Once, Include Anywhere

Author: Hans Petter Langtangen, Simula Research Laboratory and University of Oslo

Date: August 25, 2010

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

The Doconce Concept

Doconce is two things:

1. Doconce is a working strategy for documenting software in a single place and avoiding duplication of information. The slogan is: "Write once, include anywhere". This requires that what you write can be transformed to many different formats for a variety of documents (manuals, tutorials, books, doc strings, source code documentation, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. That is, the Doconce format look 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, 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 reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.

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. For example,

- 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,
- blocks of computer code can easily be included, also from source files,
- blocks of LaTeX mathematics can easily be included,
- there is support for both LaTeX and text-like inline mathematics,
- figures with captions, URLs with links, labels and references are supported,
- comments can be inserted throughout the text,
- a preprocessor (much like the C preprocessor) is integrated so other documents (files) can be included and large portions of text can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |---------------------------------|
```

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. Just a file link goes like 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 chapter ref{my:first:sec}.

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)
```

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

Another document can be included by writing `#include "mynote.do.txt"` 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)

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).

From Doconce to Other Formats

Transformation of a Doconce document to various other formats applies the script `doconce2format`:

```
Unix/DOS> doconce2format format mydoc.do.txt
```

The `preprocess` program is always used to preprocess the file first, and options to `preprocess` can be added after the filename. For example:

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections
```

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"`.

HTML

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

```
Unix/DOS> doconce2format HTML mydoc.do.txt
```

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

LaTeX

Making a LaTeX file `mydoc.tex` from `mydoc.do.txt` is done in two steps: .. Note: putting code blocks inside a list is not successful in many .. formats - the text may be messed up. A better choice is a paragraph .. environment, as used here.

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

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in a file `newcommands.tex`. If this file is present, it is 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:

```
Unix/DOS> ptex2tex mydoc
```

or just perform a plain copy:

```
Unix/DOS> cp mydoc.p.tex mydoc.tex
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. Finally, compile `mydoc.tex` the usual way and create the PDF file.

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:

```
Unix/DOS> doconce2format 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`:

```
Unix/DOS> doconce2format rst mydoc.do.txt
```

We may now produce various other formats:

```
Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
Unix/DOS> 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. That is, one can easily go from Doconce to Microsoft Word.

Sphinx

Sphinx documents can be created from a Doconce source in a few steps.

Step 1. Translate Doconce into the Sphinx dialect of the reStructuredText format:

```
Unix/DOS> doconce2format sphinx mydoc.do.txt
```

Step 2. Create a Sphinx root directory with a `conf.py` file, 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
```

Step 3. Move the `tutorial.rst` file to the Sphinx root directory:

```
Unix/DOS> mv mydoc.rst sphinx-rootdir
```

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
```

Many other formats are also possible.
Step 6. View the result:

```
Unix/DOS> firefox _build/html/index.html
```

Google Code Wiki

There are several different wiki dialects, but Doconce only support the one used by Google Code. The transformation to this format, called `gwiki` to explicitly mark it as the Google Code dialect, is done by:

```
Unix/DOS> doconce2format gwiki mydoc.do.txt
```

You can then open a new wiki page for your Google Code project, copy the `mydoc.gwiki` output file from `doconce2format` and paste the file contents into the wiki page. Press **Preview** or **Save Page** to see the formatted result.

Demos

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

```
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

Doconce needs the Python packages docutils, preprocess, and ptex2tex. The latter is only needed for the LaTeX formats.

The Doconce Documentation Strategy for User Manuals

Doconce was particularly made for writing tutorials or user manuals associated with computer codes. The text is written in Doconce format in separate files. LaTeX, HTML, XML, and other versions of the text is easily produced by the `doconce2format` script and standard tools. A plain text version is often wanted for the computer source code, this is easy to make, and then one can use `#include` statements in the computer source code to automatically get the manual or tutorial text in comments or doc strings. Below is a worked example.

Consider an example involving a Python module in a `basename.p.py` file. The `.p.py` extension identifies this as a file that has to be preprocessed) by the `preprocess` program. In a doc string in `basename.p.py` we do a preprocessor include in a comment line, say:

```
#    #include "docstrings/doc1.dst.txt
```

The file `docstrings/doc1.dst.txt` is a file filtered to a specific format (typically plain text, reStructedText, or Epytext) from an original "singleton" documentation file named `docstrings/doc1.do.txt`. The `.dst.txt` is the extension of a file filtered ready for being included in a doc string (`d` for doc, `st` for string).

For making an Epydoc manual, the `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.epytext` and renamed to `docstrings/doc1.dst.txt`. Then we run the preprocessor on the `basename.p.py` file and create a real Python file `basename.py`. Finally, we run Epydoc on this file. Alternatively, and nowadays preferably, we use Sphinx for API documentation and then the Doconce `docstrings/doc1.do.txt`

file is filtered to `docstrings/doc1.rst` and renamed to `docstrings/doc1.dst.txt`. A Sphinx directory must have been made with the right `index.rst` and `conf.py` files. Going to this directory and typing `make html` makes the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For this purpose we filter `docstrings/doc1.do.txt` to plain text format (`docstrings/doc1.txt`) and rename to `docstrings/doc1.dst.txt`. The preprocessor transforms the `basename.p.py` file to a standard Python file `basename.py`. The doc strings are now in plain text and well suited for Pydoc or reading by humans. All these steps are automated by the `insertdocstr.py` script. Here are the corresponding Unix commands:

```
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
```

Warning/Disclaimer

Doconce can be viewed is a unified interface to a variety of typesetting formats. This interface is minimal in the sense that a lot of typesetting features are not supported, for

example, footnotes and bibliography. For many documents the simple Doconce format is sufficient, while in other cases you need more sophisticated formats. Then you can just filter the Doconce text to a more approprite format and continue working in this format only. For example, reStructuredText is a good alternative: it is more tagged than Doconce and cannot be filtered to plain, untagged text, or wiki, and the LaTeX output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

Doconce Tutorial Documentation

Release 1.0

H. P. Langtangen

August 27, 2010

CONTENTS

Contents:

CHAPTER

ONE

DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

Author Hans Petter Langtangen, Simula Research Laboratory and University of Oslo

Date August 25, 2010

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

1.1 The Doconce Concept

Doconce is two things:

1. Doconce is a working strategy for documenting software in a single place and avoiding duplication of information. The slogan is: "Write once, include anywhere". This requires that what you write can be transformed to many different formats for a variety of documents (manuals, tutorials, books, doc strings, source code documentation, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. That is, the Doconce format look 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, 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 reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.

1.2 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. For example,

- 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,
- blocks of computer code can easily be included, also from source files,
- blocks of LaTeX mathematics can easily be included,
- there is support for both LaTeX and text-like inline mathematics,

- figures with captions, URLs with links, labels and references are supported,
- comments can be inserted throughout the text,
- a preprocessor (much like the C preprocessor) is integrated so other documents (files) can be included and large portions of text can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|
```

The Doconce text above results in the following little document:

1.3 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. Just a file link goes like 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 chapter ref{my:first:sec}.

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

1.4 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:

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

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)
```

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

Another document can be included by writing `#include "mynote.do.txt"` 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.

1.5 Macros (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).

CHAPTER

TWO

FROM DOCONCE TO OTHER FORMATS

Transformation of a Doconce document to various other formats applies the script `doconce2format`:

```
Unix/DOS> doconce2format format mydoc.do.txt
```

The `preprocess` program is always used to preprocess the file first, and options to `preprocess` can be added after the filename. For example,

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections
```

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"`.

2.1 HTML

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

```
Unix/DOS> doconce2format HTML mydoc.do.txt
```

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

2.2 LaTeX

Making a LaTeX file `mydoc.tex` from `mydoc.do.txt` is done in two steps: .. Note: putting code blocks inside a list is not successful in many .. formats - the text may be messed up. A better choice is a paragraph .. environment, as used here.

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

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in a file `newcommands.tex`. If this file is present, it is 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,

```
Unix/DOS> ptex2tex mydoc
```

or just perform a plain copy,

```
Unix/DOS> cp mydoc.p.tex mydoc.tex
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. Finally, compile `mydoc.tex` the usual way and create the PDF file.

2.3 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:

```
Unix/DOS> doconce2format plain mydoc.do.txt  # results in mydoc.txt
```

2.4 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`:

```
Unix/DOS> doconce2format rst mydoc.do.txt
```

We may now produce various other formats:

```
Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
Unix/DOS> 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. That is, one can easily go from Doconce to Microsoft Word.

2.5 Sphinx

Sphinx documents can be created from a Doconce source in a few steps.

Step 1. Translate Doconce into the Sphinx dialect of the reStructuredText format:

```
Unix/DOS> doconce2format sphinx mydoc.do.txt
```

Step 2. Create a Sphinx root directory with a `conf.py` file, 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
```

Step 3. Move the `tutorial.rst` file to the Sphinx root directory:

```
Unix/DOS> mv mydoc.rst sphinx-rootdir
```

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
```

Many other formats are also possible.

Step 6. View the result:

```
Unix/DOS> firefox _build/html/index.html
```

2.6 Google Code Wiki

There are several different wiki dialects, but Doconce only support the one used by Google Code. The transformation to this format, called `gwiki` to explicitly mark it as the Google Code dialect, is done by

```
Unix/DOS> doconce2format gwiki mydoc.do.txt
```

You can then open a new wiki page for your Google Code project, copy the `mydoc.gwiki` output file from `doconce2format` and paste the file contents into the wiki page. Press **Preview** or **Save Page** to see the formatted result.

2.6.1 Demos

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

```
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.

2.6.2 Dependencies

Doconce needs the Python packages docutils, preprocess, and ptex2tex. The latter is only needed for the LaTeX formats.

2.7 The Doconce Documentation Strategy for User Manuals

Doconce was particularly made for writing tutorials or user manuals associated with computer codes. The text is written in Doconce format in separate files. LaTeX, HTML, XML, and other versions of the text is easily produced by the `doconce2format` script and standard tools. A plain text version is often wanted for the computer source code, this is easy to make, and then one can use `#include` statements in the computer source code to automatically get the manual or tutorial text in comments or doc strings. Below is a worked example.

Consider an example involving a Python module in a `basename.p.py` file. The `.p.py` extension identifies this as a file that has to be preprocessed) by the `preprocess` program. In a doc string in `basename.p.py` we do a preprocessor include in a comment line, say

```
#    #include "docstrings/doc1.dst.txt
```

The file `docstrings/doc1.dst.txt` is a file filtered to a specific format (typically plain text, reStructedText, or Epytext) from an original "singleton" documentation file named `docstrings/doc1.do.txt`. The `.dst.txt` is the extension of a file filtered ready for being included in a doc string (`d` for doc, `st` for string).

For making an Epydoc manual, the `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.epytext` and renamed to `docstrings/doc1.dst.txt`. Then we run the preprocessor on the `basename.p.py` file and create a real Python file `basename.py`. Finally, we run Epydoc on this file. Alternatively, and nowadays preferably, we use Sphinx for API documentation and then the Doconce `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.rst` and renamed to `docstrings/doc1.dst.txt`. A Sphinx directory must have been made with the right `index.rst` and `conf.py` files. Going to this directory and typing `make html` makes the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For this purpose we filter `docstrings/doc1.do.txt` to plain text format (`docstrings/doc1.txt`) and rename to `docstrings/doc1.dst.txt`. The preprocessor transforms the `basename.p.py` file to a standard Python file `basename.py`. The doc strings are now in plain text and well suited for Pydoc or reading by humans. All these steps are automated by the `insertdocstr.py` script. Here are the corresponding Unix commands:

```
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
```

```
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
```

CHAPTER

THREE

WARNING/DISCLAIMER

Doconce can be viewed is a unified interface to a variety of typesetting formats. This interface is minimal in the sense that a lot of typesetting features are not supported, for example, footnotes and bibliography. For many documents the simple Doconce format is sufficient, while in other cases you need more sophisticated formats. Then you can just filter the Doconce text to a more approprite format and continue working in this format only. For example, reStructuredText is a good alternative: it is more tagged than Doconce and cannot be filtered to plain, untagged text, or wiki, and the LaTeX output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

CHAPTER

FOUR

INDICES AND TABLES

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

tutorial.txt

```
TITLE: Doconce: Document Once, Include Anywhere
AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of O
slo
DATE: August 25, 2010



 * 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 at some later stage eventually go
   with a particular format?
 * Do you find it problematic that 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
   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 working strategy for documenting software in a single
    place and avoiding duplication of information. The slogan is:
    "Write once, include anywhere". This requires that what you write
    can be transformed to many different formats for a variety of
    documents (manuals, tutorials, books, doc strings, source code
    documentation, etc.).
 2. Doconce is a simple and minimally tagged markup language that can
    be used for the above purpose. That is, the Doconce format look
    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, 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 reStructuredText
    you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the
    latter to RTF and MS Word.
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. For example,

  * 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,
  * blocks of computer code can easily be included, also from source files,
  * blocks of LaTeX mathematics can easily be included,
  * there is support for both LaTeX and text-like inline mathematics,
  * figures with captions, URLs with links, labels and references
    are supported,
  * comments can be inserted throughout the text,
  * a preprocessor (much like the C preprocessor) is integrated so
```

tutorial.txt

```
    other documents (files) can be included and large portions of text
    can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
        Just a file link goes like URL:"tutorial.do.txt". References
        to sections may use logical names as labels (e.g., a "label" command rig
ht
        after the section title), as in the reference to
        Chapter ref{my:first:sec}.

        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    |
          |--------------------------------|


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).
Just a file link goes like tutorial.do.txt. References
```

tutorial.txt

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

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::

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

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



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

```
avoiding copying information!).

Another document can be included by writing #include "mynote.do.txt"
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)
--------------------

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::

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).




From Doconce to Other Formats
=============================

Transformation of a Doconce document to various other
formats applies the script doconce2format::

        Unix/DOS> doconce2format format mydoc.do.txt


The preprocess program is always used to preprocess the file first,
and options to preprocess can be added after the filename. For example::

        Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections


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".


HTML
----
```

tutorial.txt

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

        Unix/DOS> doconce2format HTML mydoc.do.txt


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

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
     ptex2tex::

        Unix/DOS> doconce2format LaTeX mydoc.do.txt


LaTeX-specific commands ("newcommands") in math formulas and similar
can be placed in a file newcommands.tex. If this file is present,
it is 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::

        Unix/DOS> ptex2tex mydoc


or just perform a plain copy::

        Unix/DOS> cp mydoc.p.tex mydoc.tex


The ptex2tex tool makes it possible to easily switch between many
different fancy formattings of computer or verbatim code in LaTeX
documents.
Finally, compile mydoc.tex the usual way and create the PDF file.

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::

        Unix/DOS> doconce2format 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::

        Unix/DOS> doconce2format rst mydoc.do.txt
```

tutorial.txt

```
We may now produce various other formats::

        Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
        Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
        Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
        Unix/DOS> 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.
That is, one can easily go from Doconce to Microsoft Word.

Sphinx
------

Sphinx documents can be created from a Doconce source in a few steps.

*Step 1.* Translate Doconce into the Sphinx dialect of
the reStructuredText format::

        Unix/DOS> doconce2format sphinx mydoc.do.txt



*Step 2.* Create a Sphinx root directory with a conf.py file,
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



*Step 3.* Move the tutorial.rst file to the Sphinx root directory::

        Unix/DOS> mv mydoc.rst sphinx-rootdir
```

```
*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


Many other formats are also possible.

*Step 6.* View the result::

        Unix/DOS> firefox _build/html/index.html



Google Code Wiki
----------------

There are several different wiki dialects, but Doconce only support the
one used by Google Code (http://code.google.com/p/support/wiki/WikiSyntax).
The transformation to this format, called gwiki to explicitly mark
it as the Google Code dialect, is done by::

        Unix/DOS> doconce2format gwiki mydoc.do.txt


You can then open a new wiki page for your Google Code project, copy
the mydoc.gwiki output file from doconce2format and paste the
file contents into the wiki page. Press _Preview_ or _Save Page_ to
see the formatted result.



Demos
~~~~~

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

        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/trunk/docs/demo/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
~~~~~~~~~~~~

Doconce needs the Python packages
docutils (http://docutils.sourceforge.net),
preprocess (http://code.google.com/p/preprocess), and
ptex2tex (http://code.google.com/p/ptex2tex). The latter is only
needed for the LaTeX formats.


The Doconce Documentation Strategy for User Manuals
---------------------------------------------------

Doconce was particularly made for writing tutorials or user manuals
associated with computer codes. The text is written in Doconce format
in separate files. LaTeX, HTML, XML, and other versions of the text
is easily produced by the doconce2format script and standard tools.
A plain text version is often wanted for the computer source code,
this is easy to make, and then one can use
#include statements in the computer source code to automatically
get the manual or tutorial text in comments or doc strings.
Below is a worked example.

Consider an example involving a Python module in a basename.p.py file.
The .p.py extension identifies this as a file that has to be
preprocessed) by the preprocess program.
In a doc string in basename.p.py we do a preprocessor include
in a comment line, say::

        #    #include "docstrings/doc1.dst.txt


The file docstrings/doc1.dst.txt is a file filtered to a specific format
(typically plain text, reStructedText, or Epytext) from an original
"singleton" documentation file named docstrings/doc1.do.txt. The .dst.txt
is the extension of a file filtered ready for being included in a doc
string (d for doc, st for string).

For making an Epydoc manual, the docstrings/doc1.do.txt file is
filtered to docstrings/doc1.epytext and renamed to
docstrings/doc1.dst.txt.  Then we run the preprocessor on the
basename.p.py file and create a real Python file
basename.py. Finally, we run Epydoc on this file. Alternatively, and
nowadays preferably, we use Sphinx for API documentation and then the
Doconce docstrings/doc1.do.txt file is filtered to
docstrings/doc1.rst and renamed to docstrings/doc1.dst.txt. A
Sphinx directory must have been made with the right index.rst and
conf.py files. Going to this directory and typing make html makes
the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For
```

tutorial.txt

```
this purpose we filter docstrings/doc1.do.txt to plain text format
(docstrings/doc1.txt) and rename to docstrings/doc1.dst.txt. The
preprocessor transforms the basename.p.py file to a standard Python
file basename.py. The doc strings are now in plain text and well
suited for Pydoc or reading by humans. All these steps are automated
by the insertdocstr.py script.  Here are the corresponding Unix
commands::

        # make Epydoc API manual of basename module:
        cd docstrings
        doconce2format epytext doc1.do.txt
        mv doc1.epytext doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py
        epydoc basename

        # make Sphinx API manual of basename module:
        cd doc
        doconce2format sphinx doc1.do.txt
        mv doc1.rst doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py
        cd docstrings/sphinx-rootdir  # sphinx directory for API source
        make clean
        make html
        cd ../..

        # make ordinary Python module files with doc strings:
        cd docstrings
        doconce2format plain doc1.do.txt
        mv doc1.txt doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py

        # can automate inserting doc strings in all .p.py files:
        insertdocstr.py plain .
        # (runs through all .do.txt files and filters them to plain format and
        # renames to .dst.txt extension, then the script runs through all
        # .p.py files and runs the preprocessor, which includes the .dst.txt
        # files)



Warning/Disclaimer
==================

Doconce can be viewed is a unified interface to a variety of
typesetting formats.  This interface is minimal in the sense that a
lot of typesetting features are not supported, for example, footnotes
and bibliography. For many documents the simple Doconce format is
sufficient, while in other cases you need more sophisticated
formats. Then you can just filter the Doconce text to a more
approprite format and continue working in this format only.  For
example, reStructuredText is a good alternative: it is more tagged
than Doconce and cannot be filtered to plain, untagged text, or wiki,
and the LaTeX output is not at all as clean, but it also has a lot
more typesetting and tagging features than Doconce.
```

tutorial.epytext

```
TITLE: Doconce: Document Once, Include Anywhere
BY: Hans Petter Langtangen, Simula Research Laboratory and University of Oslo
DATE: August 25, 2010
 − 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 at some later stage eventually go
   with a particular format?
 − Do you find it problematic that 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
   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 working strategy for documenting software in a single
    place and avoiding duplication of information. The slogan is:
    "Write once, include anywhere". This requires that what you write
    can be transformed to many different formats for a variety of
    documents (manuals, tutorials, books, doc strings, source code
    documentation, etc.).
 2. Doconce is a simple and minimally tagged markup language that can
    be used for the above purpose. That is, the Doconce format look
    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, 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 reStructuredText
    you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the
    latter to RTF and MS Word.
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. For example,

  − 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 verbatim,
  − blocks of computer code can easily be included, also from source files,
  − blocks of LaTeX mathematics can easily be included,
  − there is support for both LaTeX and text−like inline mathematics,
  − figures with captions, URLs with links, labels and references
    are supported,
  − comments can be inserted throughout the text,
  − a preprocessor (much like the C preprocessor) is integrated so
    other documents (files) can be included and large portions of text
    can be defined in or out of the text.
Here is an example of some simple text written in the Doconce format::
```

tutorial.epytext

```
        ===== 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 http://folk.uio.no/hpl<hpl>.
        Just a file link goes like URL:"tutorial.do.txt". References
        to sections may use logical names as labels (e.g., a "label" command rig
ht
        after the section title), as in the reference to
        Chapter ref{my:first:sec}.

        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    |
          |--------------------------------|


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>}.
Just a file link goes like 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 chapter ref{my:first:sec}.
```

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)
```

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

```
Another document can be included by writing C{#include "mynote.do.txt"}
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)
--------------------

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).




From Doconce to Other Formats
=============================

Transformation of a Doconce document to various other
formats applies the script C{doconce2format}:
!bc
        Unix/DOS> doconce2format format mydoc.do.txt


The C{preprocess} program is always used to preprocess the file first,
and options to C{preprocess} can be added after the filename. For example::

        Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections


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"}.


HTML
----

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

```
is performed by::

        Unix/DOS> doconce2format HTML mydoc.do.txt


The resulting file C{mydoc.html} can be loaded into any web browser for viewing.

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
     C{ptex2tex}:
!bc
        Unix/DOS> doconce2format LaTeX mydoc.do.txt


LaTeX-specific commands ("newcommands") in math formulas and similar
can be placed in a file C{newcommands.tex}. If this file is present,
it is 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::

        Unix/DOS> ptex2tex mydoc


or just perform a plain copy::

        Unix/DOS> cp mydoc.p.tex mydoc.tex


The C{ptex2tex} tool makes it possible to easily switch between many
different fancy formattings of computer or verbatim code in LaTeX
documents.
Finally, compile C{mydoc.tex} the usual way and create the PDF file.

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::

        Unix/DOS> doconce2format 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}:
!bc
        Unix/DOS> doconce2format rst mydoc.do.txt
```

```
We may now produce various other formats::

        Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
        Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
        Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
        Unix/DOS> 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.
That is, one can easily go from Doconce to Microsoft Word.

Sphinx
------

Sphinx documents can be created from a Doconce source in a few steps.

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

        Unix/DOS> doconce2format sphinx mydoc.do.txt



I{Step 2.} Create a Sphinx root directory with a C{conf.py} file,
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



I{Step 3.} Move the C{tutorial.rst} file to the Sphinx root directory::

        Unix/DOS> mv mydoc.rst sphinx-rootdir
```

```
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


Many other formats are also possible.

I{Step 6.} View the result::

        Unix/DOS> firefox _build/html/index.html



Google Code Wiki
~~~~~~~~~~~~~~~~

There are several different wiki dialects, but Doconce only support the
one used by U{Google Code<http://code.google.com/p/support/wiki/WikiSyntax>}.
The transformation to this format, called C{gwiki} to explicitly mark
it as the Google Code dialect, is done by::

        Unix/DOS> doconce2format gwiki mydoc.do.txt


You can then open a new wiki page for your Google Code project, copy
the C{mydoc.gwiki} output file from C{doconce2format} and paste the
file contents into the wiki page. Press B{Preview} or B{Save Page} to
see the formatted result.



Demos
~~~~~

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

        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.
```

tutorial.epytext

```
U{Here<https://doconce.googlecode.com/hg/trunk/docs/demo/index.html>} is a sampl
e
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
~~~~~~~~~~~~

Doconce needs the Python packages
U{docutils<http://docutils.sourceforge.net>},
U{preprocess<http://code.google.com/p/preprocess>}, and
U{ptex2tex<http://code.google.com/p/ptex2tex>}. The latter is only
needed for the LaTeX formats.


The Doconce Documentation Strategy for User Manuals
---------------------------------------------------

Doconce was particularly made for writing tutorials or user manuals
associated with computer codes. The text is written in Doconce format
in separate files. LaTeX, HTML, XML, and other versions of the text
is easily produced by the C{doconce2format} script and standard tools.
A plain text version is often wanted for the computer source code,
this is easy to make, and then one can use
C{#include} statements in the computer source code to automatically
get the manual or tutorial text in comments or doc strings.
Below is a worked example.

Consider an example involving a Python module in a C{basename.p.py} file.
The C{.p.py} extension identifies this as a file that has to be
preprocessed) by the C{preprocess} program.
In a doc string in C{basename.p.py} we do a preprocessor include
in a comment line, say::

        #    #include "docstrings/doc1.dst.txt


The file C{docstrings/doc1.dst.txt} is a file filtered to a specific format
(typically plain text, reStructedText, or Epytext) from an original
"singleton" documentation file named C{docstrings/doc1.do.txt}. The C{.dst.txt}
is the extension of a file filtered ready for being included in a doc
string (C{d} for doc, C{st} for string).

For making an Epydoc manual, the C{docstrings/doc1.do.txt} file is
filtered to C{docstrings/doc1.epytext} and renamed to
C{docstrings/doc1.dst.txt}.  Then we run the preprocessor on the
C{basename.p.py} file and create a real Python file
C{basename.py}. Finally, we run Epydoc on this file. Alternatively, and
nowadays preferably, we use Sphinx for API documentation and then the
Doconce C{docstrings/doc1.do.txt} file is filtered to
C{docstrings/doc1.rst} and renamed to C{docstrings/doc1.dst.txt}. A
Sphinx directory must have been made with the right C{index.rst} and
C{conf.py} files. Going to this directory and typing C{make html} makes
the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For
```

```
this purpose we filter C{docstrings/doc1.do.txt} to plain text format
(C{docstrings/doc1.txt}) and rename to C{docstrings/doc1.dst.txt}. The
preprocessor transforms the C{basename.p.py} file to a standard Python
file C{basename.py}. The doc strings are now in plain text and well
suited for Pydoc or reading by humans. All these steps are automated
by the C{insertdocstr.py} script.  Here are the corresponding Unix
commands::

        # make Epydoc API manual of basename module:
        cd docstrings
        doconce2format epytext doc1.do.txt
        mv doc1.epytext doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py
        epydoc basename

        # make Sphinx API manual of basename module:
        cd doc
        doconce2format sphinx doc1.do.txt
        mv doc1.rst doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py
        cd docstrings/sphinx-rootdir  # sphinx directory for API source
        make clean
        make html
        cd ../..

        # make ordinary Python module files with doc strings:
        cd docstrings
        doconce2format plain doc1.do.txt
        mv doc1.txt doc1.dst.txt
        cd ..
        preprocess basename.p.py > basename.py

        # can automate inserting doc strings in all .p.py files:
        insertdocstr.py plain .
        # (runs through all .do.txt files and filters them to plain format and
        # renames to .dst.txt extension, then the script runs through all
        # .p.py files and runs the preprocessor, which includes the .dst.txt
        # files)



Warning/Disclaimer
==================

Doconce can be viewed is a unified interface to a variety of
typesetting formats.  This interface is minimal in the sense that a
lot of typesetting features are not supported, for example, footnotes
and bibliography. For many documents the simple Doconce format is
sufficient, while in other cases you need more sophisticated
formats. Then you can just filter the Doconce text to a more
approprite format and continue working in this format only.  For
example, reStructuredText is a good alternative: it is more tagged
than Doconce and cannot be filtered to plain, untagged text, or wiki,
and the LaTeX output is not at all as clean, but it also has a lot
more typesetting and tagging features than Doconce.
```

tutorial.gwiki

```
#summary Doconce: Document Once, Include Anywhere
<wiki:toc max_depth="2" />

==== Hans Petter Langtangen, Simula Research Laboratory and University of Oslo =
===

==== August 25, 2010 ====

<wiki:comment> lines beginning with # are comment lines </wiki:comment>



 * 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 at some later stage eventually go
 with a particular format?
 * Do you find it problematic that you have the same information   scattered aro
und in different documents in different typesetting   formats? Would it be a goo
d idea to write things once, 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 working strategy for documenting software in a single    place a
nd avoiding duplication of information. The slogan is:    "Write once, include a
nywhere". This requires that what you write    can be transformed to many differ
ent formats for a variety of    documents (manuals, tutorials, books, doc string
s, source code    documentation, etc.).
 # Doconce is a simple and minimally tagged markup language that can    be used
for the above purpose. That is, the Doconce format look    like ordinary ASCII t
ext (much like what you would use in an    email), but the text can be transform
ed to numerous other formats,    including HTML, Wiki, LaTeX, PDF, reStructuredT
ext (reST), Sphinx,    Epytext, and also plain text (where non-obvious formattin
g/tags are    removed for clear reading in, e.g., emails). From reStructuredText
    you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the    latter to R
TF and MS Word.

==== 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. For example,



  * 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 ve
rbatim,
  * blocks of computer code can easily be included, also from source files,
  * blocks of LaTeX mathematics can easily be included,
  * there is support for both LaTeX and text-like inline mathematics,
```

```
  * figures with captions, URLs with links, labels and references    are support
ed,
  * comments can be inserted throughout the text,
  * a preprocessor (much like the C preprocessor) is integrated so    other docu
ments (files) can be included and large portions of text    can be defined in or
 out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|
}}}
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:
```

tutorial.gwiki

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

URLs with a link word are possible, as in [http://folk.uio.no/hpl hpl].
Just a file link goes like 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 chapter ref{my:first:sec}.

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)
}}}

One can also copy computer code directly from files, either the
```

```
complete file or specified parts.  Computer code is then never
duplicated in the documentation (important for the principle of
avoiding copying information!).

Another document can be included by writing ‘#include "mynote.do.txt"‘
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) ====

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).



<wiki:comment> Example on including another Doconce file: </wiki:comment>



== From Doconce to Other Formats ==

Transformation of a Doconce document to various other
formats applies the script ‘doconce2format‘:
{{{
Unix/DOS> doconce2format format mydoc.do.txt
}}}
The ‘preprocess‘ program is always used to preprocess the file first,
and options to ‘preprocess‘ can be added after the filename. For example,
{{{
Unix/DOS> doconce2format LaTeX mydoc.do.txt −Dextra_sections
}}}
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"‘.

==== HTML ====

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

```
Unix/DOS> doconce2format HTML mydoc.do.txt
}}}
The resulting file ‘mydoc.html‘ can be loaded into any web browser for viewing.

==== 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
     ‘ptex2tex‘:
{{{
Unix/DOS> doconce2format LaTeX mydoc.do.txt
}}}
LaTeX−specific commands ("newcommands") in math formulas and similar
can be placed in a file ‘newcommands.tex‘. If this file is present,
it is 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,
{{{
Unix/DOS> ptex2tex mydoc
}}}
or just perform a plain copy,
{{{
Unix/DOS> cp mydoc.p.tex mydoc.tex
}}}
The ‘ptex2tex‘ tool makes it possible to easily switch between many
different fancy formattings of computer or verbatim code in LaTeX
documents.
Finally, compile ‘mydoc.tex‘ the usual way and create the PDF file.

==== 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:
{{{
Unix/DOS> doconce2format 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‘:
{{{
Unix/DOS> doconce2format rst mydoc.do.txt
}}}
We may now produce various other formats:
{{{
Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
Unix/DOS> 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.
That is, one can easily go from Doconce to Microsoft Word.

==== Sphinx ====

Sphinx documents can be created from a Doconce source in a few steps.

*Step 1.* Translate Doconce into the Sphinx dialect of
the reStructuredText format:
{{{
Unix/DOS> doconce2format sphinx mydoc.do.txt
}}}

*Step 2.* Create a Sphinx root directory with a ‘conf.py‘ file,
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
}}}

*Step 3.* Move the ‘tutorial.rst‘ file to the Sphinx root directory:
{{{
Unix/DOS> mv mydoc.rst sphinx-rootdir
}}}

*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!)
```

tutorial.gwiki

```
*Step 5.* Generate, for instance, an HTML version of the Sphinx source:
{{{
make clean   # remove old versions
make html
}}}
Many other formats are also possible.

*Step 6.* View the result:
{{{
Unix/DOS> firefox _build/html/index.html
}}}

==== Google Code Wiki ====

There are several different wiki dialects, but Doconce only support the
one used by [http://code.google.com/p/support/wiki/WikiSyntax Google Code].
The transformation to this format, called ‘gwiki‘ to explicitly mark
it as the Google Code dialect, is done by
{{{
Unix/DOS> doconce2format gwiki mydoc.do.txt
}}}
You can then open a new wiki page for your Google Code project, copy
the ‘mydoc.gwiki‘ output file from ‘doconce2format‘ and paste the
file contents into the wiki page. Press *Preview* or *Save Page* to
see the formatted result.

==== Demos ====

The current text is generated from a Doconce format stored in the file
{{{
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/trunk/docs/demo/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 ====

Doconce needs the Python packages
[http://docutils.sourceforge.net docutils],
[http://code.google.com/p/preprocess preprocess], and
[http://code.google.com/p/ptex2tex ptex2tex]. The latter is only
needed for the LaTeX formats.

==== The Doconce Documentation Strategy for User Manuals ====

Doconce was particularly made for writing tutorials or user manuals
associated with computer codes. The text is written in Doconce format
in separate files. LaTeX, HTML, XML, and other versions of the text
```

```
is easily produced by the ‘doconce2format‘ script and standard tools.
A plain text version is often wanted for the computer source code,
this is easy to make, and then one can use
‘#include‘ statements in the computer source code to automatically
get the manual or tutorial text in comments or doc strings.
Below is a worked example.

Consider an example involving a Python module in a ‘basename.p.py‘ file.
The ‘.p.py‘ extension identifies this as a file that has to be
preprocessed) by the ‘preprocess‘ program.
In a doc string in ‘basename.p.py‘ we do a preprocessor include
in a comment line, say
{{{
#    #include "docstrings/doc1.dst.txt
}}}
<wiki:comment>  </wiki:comment>
<wiki:comment> Note: we insert an error right above as the right quote is missin
g. </wiki:comment>
<wiki:comment> Then preprocess skips the statement, otherwise it gives an error
</wiki:comment>
<wiki:comment> message about a missing file docstrings/doc1.dst.txt (which we do
n’t </wiki:comment>
<wiki:comment> have, it’s just a sample file name). Also note that comment lines
 </wiki:comment>
<wiki:comment> must not come before a code block for the rst/st/epytext formats
to work. </wiki:comment>
<wiki:comment>  </wiki:comment>
The file ‘docstrings/doc1.dst.txt‘ is a file filtered to a specific format
(typically plain text, reStructedText, or Epytext) from an original
"singleton" documentation file named ‘docstrings/doc1.do.txt‘. The ‘.dst.txt‘
is the extension of a file filtered ready for being included in a doc
string (‘d‘ for doc, ‘st‘ for string).

For making an Epydoc manual, the ‘docstrings/doc1.do.txt‘ file is
filtered to ‘docstrings/doc1.epytext‘ and renamed to
‘docstrings/doc1.dst.txt‘.  Then we run the preprocessor on the
‘basename.p.py‘ file and create a real Python file
‘basename.py‘. Finally, we run Epydoc on this file. Alternatively, and
nowadays preferably, we use Sphinx for API documentation and then the
Doconce ‘docstrings/doc1.do.txt‘ file is filtered to
‘docstrings/doc1.rst‘ and renamed to ‘docstrings/doc1.dst.txt‘. A
Sphinx directory must have been made with the right ‘index.rst‘ and
‘conf.py‘ files. Going to this directory and typing ‘make html‘ makes
the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For
this purpose we filter ‘docstrings/doc1.do.txt‘ to plain text format
(‘docstrings/doc1.txt‘) and rename to ‘docstrings/doc1.dst.txt‘. The
preprocessor transforms the ‘basename.p.py‘ file to a standard Python
file ‘basename.py‘. The doc strings are now in plain text and well
suited for Pydoc or reading by humans. All these steps are automated
by the ‘insertdocstr.py‘ script.  Here are the corresponding Unix
commands:
{{{
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
```

```
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
}}}



== Warning/Disclaimer ==

Doconce can be viewed is a unified interface to a variety of
typesetting formats.  This interface is minimal in the sense that a
lot of typesetting features are not supported, for example, footnotes
and bibliography. For many documents the simple Doconce format is
sufficient, while in other cases you need more sophisticated
formats. Then you can just filter the Doconce text to a more
approprite format and continue working in this format only.  For
example, reStructuredText is a good alternative: it is more tagged
than Doconce and cannot be filtered to plain, untagged text, or wiki,
and the LaTeX output is not at all as clean, but it also has a lot
more typesetting and tagging features than Doconce.
```

Doconce Tutorial Documentation

Release 1.0

H. P. Langtangen

August 27, 2010

CONTENTS

Contents:

CHAPTER

ONE

DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

Author Hans Petter Langtangen, Simula Research Laboratory and University of Oslo

Date August 25, 2010

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

1.1 The Doconce Concept

Doconce is two things:

1. Doconce is a working strategy for documenting software in a single place and avoiding duplication of information. The slogan is: "Write once, include anywhere". This requires that what you write can be transformed to many different formats for a variety of documents (manuals, tutorials, books, doc strings, source code documentation, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. That is, the Doconce format look 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, 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 reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.

1.2 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. For example,

- 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,
- blocks of computer code can easily be included, also from source files,
- blocks of LaTeX mathematics can easily be included,
- there is support for both LaTeX and text-like inline mathematics,

- figures with captions, URLs with links, labels and references are supported,
- comments can be inserted throughout the text,
- a preprocessor (much like the C preprocessor) is integrated so other documents (files) can be included and large portions of text can be defined in or out of the 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 http://folk.uio.no/hpl<hpl>.
Just a file link goes like 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
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|
```

The Doconce text above results in the following little document:

1.3 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. Just a file link goes like 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 chapter ref{my:first:sec}.

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

1.4 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:

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

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)
```

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

Another document can be included by writing `#include "mynote.do.txt"` 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.

1.5 Macros (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).

CHAPTER

TWO

FROM DOCONCE TO OTHER FORMATS

Transformation of a Doconce document to various other formats applies the script `doconce2format`:

```
Unix/DOS> doconce2format format mydoc.do.txt
```

The `preprocess` program is always used to preprocess the file first, and options to `preprocess` can be added after the filename. For example,

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra_sections
```

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"`.

2.1 HTML

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

```
Unix/DOS> doconce2format HTML mydoc.do.txt
```

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

2.2 LaTeX

Making a LaTeX file `mydoc.tex` from `mydoc.do.txt` is done in two steps: .. Note: putting code blocks inside a list is not successful in many .. formats - the text may be messed up. A better choice is a paragraph .. environment, as used here.

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

```
Unix/DOS> doconce2format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in a file `newcommands.tex`. If this file is present, it is 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,

```
Unix/DOS> ptex2tex mydoc
```

or just perform a plain copy,

```
Unix/DOS> cp mydoc.p.tex mydoc.tex
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. Finally, compile `mydoc.tex` the usual way and create the PDF file.

2.3 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:

```
Unix/DOS> doconce2format plain mydoc.do.txt  # results in mydoc.txt
```

2.4 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`:

```
Unix/DOS> doconce2format rst mydoc.do.txt
```

We may now produce various other formats:

```
Unix/DOS> rst2html.py  mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py   mydoc.rst > mydoc.xml  # XML
Unix/DOS> 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. That is, one can easily go from Doconce to Microsoft Word.

2.5 Sphinx

Sphinx documents can be created from a Doconce source in a few steps.

Step 1. Translate Doconce into the Sphinx dialect of the reStructuredText format:

```
Unix/DOS> doconce2format sphinx mydoc.do.txt
```

Step 2. Create a Sphinx root directory with a `conf.py` file, 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
```

Step 3. Move the `tutorial.rst` file to the Sphinx root directory:

```
Unix/DOS> mv mydoc.rst sphinx-rootdir
```

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
```

Many other formats are also possible.

Step 6. View the result:

```
Unix/DOS> firefox _build/html/index.html
```

2.6 Google Code Wiki

There are several different wiki dialects, but Doconce only support the one used by Google Code. The transformation to this format, called `gwiki` to explicitly mark it as the Google Code dialect, is done by

```
Unix/DOS> doconce2format gwiki mydoc.do.txt
```

You can then open a new wiki page for your Google Code project, copy the `mydoc.gwiki` output file from `doconce2format` and paste the file contents into the wiki page. Press **Preview** or **Save Page** to see the formatted result.

2.6.1 Demos

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

```
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.

2.6.2 Dependencies

Doconce needs the Python packages docutils, preprocess, and ptex2tex. The latter is only needed for the LaTeX formats.

2.7 The Doconce Documentation Strategy for User Manuals

Doconce was particularly made for writing tutorials or user manuals associated with computer codes. The text is written in Doconce format in separate files. LaTeX, HTML, XML, and other versions of the text is easily produced by the `doconce2format` script and standard tools. A plain text version is often wanted for the computer source code, this is easy to make, and then one can use `#include` statements in the computer source code to automatically get the manual or tutorial text in comments or doc strings. Below is a worked example.

Consider an example involving a Python module in a `basename.p.py` file. The `.p.py` extension identifies this as a file that has to be preprocessed) by the `preprocess` program. In a doc string in `basename.p.py` we do a preprocessor include in a comment line, say

```
#    #include "docstrings/doc1.dst.txt
```

The file `docstrings/doc1.dst.txt` is a file filtered to a specific format (typically plain text, reStructedText, or Epytext) from an original "singleton" documentation file named `docstrings/doc1.do.txt`. The `.dst.txt` is the extension of a file filtered ready for being included in a doc string (`d` for doc, `st` for string).

For making an Epydoc manual, the `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.epytext` and renamed to `docstrings/doc1.dst.txt`. Then we run the preprocessor on the `basename.p.py` file and create a real Python file `basename.py`. Finally, we run Epydoc on this file. Alternatively, and nowadays preferably, we use Sphinx for API documentation and then the Doconce `docstrings/doc1.do.txt` file is filtered to `docstrings/doc1.rst` and renamed to `docstrings/doc1.dst.txt`. A Sphinx directory must have been made with the right `index.rst` and `conf.py` files. Going to this directory and typing `make html` makes the HTML version of the Sphinx API documentation.

The next step is to produce the final pure Python source code. For this purpose we filter `docstrings/doc1.do.txt` to plain text format (`docstrings/doc1.txt`) and rename to `docstrings/doc1.dst.txt`. The preprocessor transforms the `basename.p.py` file to a standard Python file `basename.py`. The doc strings are now in plain text and well suited for Pydoc or reading by humans. All these steps are automated by the `insertdocstr.py` script. Here are the corresponding Unix commands:

```
# make Epydoc API manual of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
```

```
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py > basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)
```

CHAPTER

THREE

WARNING/DISCLAIMER

Doconce can be viewed is a unified interface to a variety of typesetting formats. This interface is minimal in the sense that a lot of typesetting features are not supported, for example, footnotes and bibliography. For many documents the simple Doconce format is sufficient, while in other cases you need more sophisticated formats. Then you can just filter the Doconce text to a more approprite format and continue working in this format only. For example, reStructuredText is a good alternative: it is more tagged than Doconce and cannot be filtered to plain, untagged text, or wiki, and the LaTeX output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

CHAPTER

FOUR

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 0.8 -->
<document source="tutorial.rst"><section ids="doconce-document-once-include-anyw
here" names="doconce:\ document\ once,\ include\ anywhere"><title>Doconce: Docum
ent Once, Include Anywhere</title><field_list><field><field_name>Author</field_n
ame><field_body><paragraph>Hans Petter Langtangen, Simula Research Laboratory an
d University of Oslo</paragraph></field_body></field><field><field_name>Date</fi
eld_name><field_body><paragraph>August 25, 2010</paragraph></field_body></field>
</field_list><comment xml:space="preserve">lines beginning with # are comment li
nes


* 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 at some later stage eventually go
  with a particular format?

* Do you find it problematic that 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
  place, and include it anywhere?</comment><paragraph>If any of these questions
are of interest, you should keep on reading.</paragraph><section ids="the-doconc
e-concept" names="the\ doconce\ concept"><title>The Doconce Concept</title><para
graph>Doconce is two things:</paragraph><block_quote><enumerated_list enumtype="
arabic" prefix="" suffix="."><list_item><paragraph>Doconce is a working strategy
 for documenting software in a single
place and avoiding duplication of information. The slogan is:
&amp;quot;Write once, include anywhere&amp;quot;. This requires that what you write
can be transformed to many different formats for a variety of
documents (manuals, tutorials, books, doc strings, source code
documentation, etc.).</paragraph></list_item><list_item><paragraph>Doconce is a
simple and minimally tagged markup language that can
be used for the above purpose. That is, the Doconce format look
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, 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 reStructuredText
you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the
latter to RTF and MS Word.</paragraph></list_item></enumerated_list></block_quot
e></section><section ids="what-does-doconce-look-like" names="what\ does\ doconc
e\ look\ like?"><title>What Does Doconce Look Like?</title><paragraph>Doconce te
xt looks like ordinary text, but there are some almost invisible
text constructions that allow you to control the formating. For example,</paragr
aph><block_quote><bullet_list bullet="*"><list_item><paragraph>bullet lists aris
e from lines starting with an asterisk,</paragraph></list_item><list_item><parag
raph><emphasis>emphasized words</emphasis> are surrounded by asterisks,</paragra
ph></list_item><list_item><paragraph><strong>words in boldface</strong> are surr
ounded by underscores,</paragraph></list_item><list_item><paragraph>words from c
omputer code are enclosed in back quotes and
then typeset verbatim,</paragraph></list_item><list_item><paragraph>blocks of co
mputer code can easily be included, also from source files,</paragraph></list_it
em><list_item><paragraph>blocks of LaTeX mathematics can easily be included,</pa
ragraph></list_item><list_item><paragraph>there is support for both LaTeX and te
```

tutorial.xml

```
xt-like inline mathematics,</paragraph></list_item><list_item><paragraph>figures
 with captions, URLs with links, labels and references
are supported,</paragraph></list_item><list_item><paragraph>comments can be inse
rted throughout the text,</paragraph></list_item><list_item><paragraph>a preproc
essor (much like the C preprocessor) is integrated so
other documents (files) can be included and large portions of text
can be defined in or out of the text.</paragraph></list_item></bullet_list></blo
ck_quote><paragraph>Here is an example of some simple text written in the Doconc
e 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 http://folk.uio.no/hpl&lt;hpl&gt;.
Just a file link goes like URL:&quot;tutorial.do.txt&quot;. References
to sections may use logical names as labels (e.g., a &quot;label&quot; command r
ight
after the section title), as in the reference to
Chapter ref{my:first:sec}.

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    |
  |--------------------------------|</literal_block><paragraph>The Doconce text
above results in the following little document:</paragraph></section><section id
s="a-subsection-with-sample-text" names="a\ subsection\ with\ sample\ text"><tit
le>A Subsection with Sample Text</title><paragraph>Ordinary text looks like ordi
nary 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
```

tutorial.xml

```
k.uio.no/hpl">hpl</reference><target ids="hpl" names="hpl" refuri="http://folk.u
io.no/hpl"/>.
Just a file link goes like <reference name="tutorial.do.txt" refuri="tutorial.do
.txt">tutorial.do.txt</reference><target ids="tutorial-do-txt" names="tutorial.d
o.txt" refuri="tutorial.do.txt"/>. References
to sections may use logical names as labels (e.g., a &quot;label&quot; command r
ight
after the section title), as in the reference to
the chapter ref{my:first:sec}.</paragraph><paragraph>Tables are also supperted,
e.g.,</paragraph><table><tgroup cols="3"><colspec colwidth="12"/><colspec colwid
th="12"/><colspec colwidth="12"/><thead><row><entry><paragraph>time</paragraph><
/entry><entry><paragraph>velocity</paragraph></entry><entry><paragraph>accelerat
ion</paragraph></entry></row></thead><tbody><row><entry><paragraph>0.0</paragrap
h></entry><entry><paragraph>1.4186</paragraph></entry><entry><paragraph>-5.01</p
aragraph></entry></row><row><entry><paragraph>2.0</paragraph></entry><entry><par
agraph>1.376512</paragraph></entry><entry><paragraph>11.919</paragraph></entry><
/row><row><entry><paragraph>4.0</paragraph></entry><entry><paragraph>1.1E+1</par
agraph></entry><entry><paragraph>14.717624</paragraph></entry></row></tbody></tg
roup></table></section><section ids="mathematics-and-computer-code" names="mathe
matics\ and\ computer\ code"><title>Mathematics and Computer Code</title><paragr
aph>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>One can
also copy computer code directly from files, either the
complete file or specified parts.  Computer code is then never
duplicated in the documentation (important for the principle of
avoiding copying information!).</paragraph><paragraph>Another document can be in
cluded by writing <literal>#include &quot;mynote.do.txt&quot;</literal>
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
```

tutorial.xml

```
right writing enviroment for plain text.</paragraph></section><section ids="macr
os-newcommands" names="macros\ (newcommands)"><title>Macros (Newcommands)</title
><paragraph>Doconce supports a type of macros via a LaTeX-style <emphasis>newcom
mand</emphasis>
construction.  The newcommands defined in a file with name
<literal>newcommand_replace.tex</literal> are expanded when Doconce is filtered
to
other formats, except for LaTeX (since LaTeX performs the expansion
itself).  Newcommands in files with names <literal>newcommands.tex</literal> and
<literal>newcommands_keep.tex</literal> are kept unaltered when Doconce text is
filtered to other formats, except for the Sphinx format. Since Sphinx
understands LaTeX math, but not newcommands if the Sphinx output is
HTML, it makes most sense to expand all newcommands.  Normally, a user
will put all newcommands that appear in math blocks surrounded by
<literal>!bt</literal> and <literal>!et</literal> in <literal>newcommands_keep.t
ex</literal> to keep them unchanged, at
least if they contribute to make the raw LaTeX math text easier to
read in the formats that cannot render LaTeX.  Newcommands used
elsewhere throughout the text will usually be placed in
<literal>newcommands_replace.tex</literal> and expanded by Doconce.  The definit
ions of
newcommands in the <literal>newcommands*.tex</literal> files <emphasis>must</emp
hasis> appear on a single
line (multi-line newcommands are too hard to parse with regular
expressions).</paragraph><comment xml:space="preserve">Example on including anot
her Doconce file:</comment></section></section><section ids="from-doconce-to-oth
er-formats" names="from\ doconce\ to\ other\ formats"><title>From Doconce to Oth
er Formats</title><paragraph>Transformation of a Doconce document to various oth
er
formats applies the script <literal>doconce2format</literal>:</paragraph><litera
l_block xml:space="preserve">Unix/DOS&gt; doconce2format format mydoc.do.txt</li
teral_block><paragraph>The <literal>preprocess</literal> program is always used
to preprocess the file first,
and options to <literal>preprocess</literal> can be added after the filename. Fo
r example:</paragraph><literal_block xml:space="preserve">Unix/DOS&gt; doconce2f
ormat LaTeX mydoc.do.txt -Dextra_sections</literal_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 == &amp;quot;LaTeX&qu
ot;</literal>.</paragraph><section ids="html" names="html"><title>HTML</title><p
aragraph>Making an HTML version of a Doconce file <literal>mydoc.do.txt</literal
>
is performed by:</paragraph><literal_block xml:space="preserve">Unix/DOS&gt; doc
once2format HTML mydoc.do.txt</literal_block><paragraph>The resulting file <lite
ral>mydoc.html</literal> can be loaded into any web browser for viewing.</paragr
aph></section><section ids="latex" names="latex"><title>LaTeX</title><paragraph>
Making a LaTeX file <literal>mydoc.tex</literal> from <literal>mydoc.do.txt</lit
eral> is done in two steps:
.. Note: putting code blocks inside a list is not successful in many
.. formats - the text may be messed up. A better choice is a paragraph
.. environment, as used here.</paragraph><definition_list><definition_list_item>
<term><emphasis>Step 1.</emphasis> Filter the doconce text to a pre-LaTeX form <
literal>mydoc.p.tex</literal> for</term><definition><paragraph><literal>ptex2tex
</literal>:</paragraph><literal_block xml:space="preserve">Unix/DOS&gt; doconce2
format LaTeX mydoc.do.txt</literal_block></definition></definition_list_item></d
efinition_list><paragraph>LaTeX-specific commands (&amp;quot;newcommands&amp;quot;) in m
```

tutorial.xml

```
ath formulas and similar
can be placed in a file <literal>newcommands.tex</literal>. If this file is pres
ent,
it is included in the LaTeX document so that your commands are
defined.</paragraph><paragraph><emphasis>Step 2.</emphasis> Run <literal>ptex2te
x</literal> (if you have it) to make a standard LaTeX file:</paragraph><literal_
block xml:space="preserve">Unix/DOS&gt; ptex2tex mydoc</literal_block><paragraph
>or just perform a plain copy:</paragraph><literal_block xml:space="preserve">Un
ix/DOS&gt; cp mydoc.p.tex mydoc.tex</literal_block><paragraph>The <literal>ptex2
tex</literal> tool makes it possible to easily switch between many
different fancy formattings of computer or verbatim code in LaTeX
documents.
Finally, compile <literal>mydoc.tex</literal> the usual way and create the PDF f
ile.</paragraph></section><section ids="plain-ascii-text" names="plain\ ascii\ t
ext"><title>Plain ASCII Text</title><paragraph>We can go from Doconce &quot;back
 to&quot; plain untagged text suitable for viewing
in terminal windows, inclusion in email text, or for insertion in
computer source code:</paragraph><literal_block xml:space="preserve">Unix/DOS&gt
; doconce2format plain mydoc.do.txt  # results in mydoc.txt</literal_block></sec
tion><section ids="restructuredtext" names="restructuredtext"><title>reStructure
dText</title><paragraph>Going from Doconce to reStructuredText gives a lot of po
ssibilities 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">Unix/DOS&gt; doconce2format rst mydoc.do.txt</literal_block><
paragraph>We may now produce various other formats:</paragraph><literal_block xm
l:space="preserve">Unix/DOS&gt; rst2html.py  mydoc.rst &gt; mydoc.html # HTML
Unix/DOS&gt; rst2latex.py mydoc.rst &gt; mydoc.tex  # LaTeX
Unix/DOS&gt; rst2xml.py   mydoc.rst &gt; mydoc.xml  # XML
Unix/DOS&gt; rst2odt.py   mydoc.rst &gt; mydoc.odt  # OpenOffice</literal_block>
<paragraph>The OpenOffice file <literal>mydoc.odt</literal> can be loaded into O
penOffice and
saved in, among other things, the RTF format or the Microsoft Word format.
That is, one can easily go from Doconce to Microsoft Word.</paragraph></section>
<section ids="sphinx" names="sphinx"><title>Sphinx</title><paragraph>Sphinx docu
ments can be created from a Doconce source in a few steps.</paragraph><paragraph
><emphasis>Step 1.</emphasis> Translate Doconce into the Sphinx dialect of
the reStructuredText format:</paragraph><literal_block xml:space="preserve">Unix
/DOS&gt; doconce2format sphinx mydoc.do.txt</literal_block><paragraph><emphasis>
Step 2.</emphasis> Create a Sphinx root directory with a <literal>conf.py</liter
al> file,
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
```

tutorial.xml

```
n
y
n
n
y
y
y
EOF</literal_block><paragraph><emphasis>Step 3.</emphasis> Move the <literal>tut
orial.rst</literal> file to the Sphinx root directory:</paragraph><literal_block
 xml:space="preserve">Unix/DOS&gt; mv mydoc.rst sphinx-rootdir</literal_block><p
aragraph><emphasis>Step 4.</emphasis> Edit the generated <literal>index.rst</lit
eral> 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>Many other formats are also possible.</parag
raph><paragraph><emphasis>Step 6.</emphasis> View the result:</paragraph><litera
l_block xml:space="preserve">Unix/DOS&gt; firefox _build/html/index.html</litera
l_block></section><section ids="google-code-wiki" names="google\ code\ wiki"><ti
tle>Google Code Wiki</title><paragraph>There are several different wiki dialects
, but Doconce only support the
one used by <reference name="Google Code" refuri="http://code.google.com/p/suppo
rt/wiki/WikiSyntax">Google Code</reference><target ids="google-code" names="goog
le\ code" refuri="http://code.google.com/p/support/wiki/WikiSyntax"/>.
The transformation to this format, called <literal>gwiki</literal> to explicitly
 mark
it as the Google Code dialect, is done by:</paragraph><literal_block xml:space="
preserve">Unix/DOS&gt; doconce2format gwiki mydoc.do.txt</literal_block><paragra
ph>You can then open a new wiki page for your Google Code project, copy
the <literal>mydoc.gwiki</literal> output file from <literal>doconce2format</lit
eral> and paste the
file contents into the wiki page. Press <strong>Preview</strong> or <strong>Save
 Page</strong> to
see the formatted result.</paragraph><section ids="demos" names="demos"><title>D
emos</title><paragraph>The current text is generated from a Doconce format store
d in the file:</paragraph><literal_block xml:space="preserve">tutorial/tutorial.
do.txt</literal_block><paragraph>The file <literal>make.sh</literal> in the <lit
eral>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/trunk/docs/demo
/index.html">Here</reference><target ids="here" names="here" refuri="https://doc
once.googlecode.com/hg/trunk/docs/demo/index.html"/> is a sample
of how this tutorial looks in different formats.</paragraph><paragraph>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
```

tutorial.xml

```
s.</paragraph></section><section ids="dependencies" names="dependencies"><title>
Dependencies</title><paragraph>Doconce needs the Python packages
<reference name="docutils" refuri="http://docutils.sourceforge.net">docutils</re
ference><target ids="docutils" names="docutils" refuri="http://docutils.sourcefo
rge.net"/>,
<reference name="preprocess" refuri="http://code.google.com/p/preprocess">prepro
cess</reference><target ids="preprocess" names="preprocess" refuri="http://code.
google.com/p/preprocess"/>, and
<reference name="ptex2tex" refuri="http://code.google.com/p/ptex2tex">ptex2tex</
reference><target ids="ptex2tex" names="ptex2tex" refuri="http://code.google.com
/p/ptex2tex"/>. The latter is only
needed for the LaTeX formats.</paragraph></section></section><section ids="the-d
oconce-documentation-strategy-for-user-manuals" names="the\ doconce\ documentati
on\ strategy\ for\ user\ manuals"><title>The Doconce Documentation Strategy for
User Manuals</title><paragraph>Doconce was particularly made for writing tutoria
ls or user manuals
associated with computer codes. The text is written in Doconce format
in separate files. LaTeX, HTML, XML, and other versions of the text
is easily produced by the <literal>doconce2format</literal> script and standard
tools.
A plain text version is often wanted for the computer source code,
this is easy to make, and then one can use
<literal>#include</literal> statements in the computer source code to automatica
lly
get the manual or tutorial text in comments or doc strings.
Below is a worked example.</paragraph><paragraph>Consider an example involving a
 Python module in a <literal>basename.p.py</literal> file.
The <literal>.p.py</literal> extension identifies this as a file that has to be
preprocessed) by the <literal>preprocess</literal> program.
In a doc string in <literal>basename.p.py</literal> we do a preprocessor include
in a comment line, say:</paragraph><literal_block xml:space="preserve">#    #inc
lude &amp;quot;docstrings/doc1.dst.txt</literal_block><comment xml:space="preserve">
Note: we insert an error right above as the right quote is missing.</comment><co
mment xml:space="preserve">Then preprocess skips the statement, otherwise it giv
es an error</comment><comment xml:space="preserve">message about a missing file
docstrings/doc1.dst.txt (which we don't</comment><comment xml:space="preserve">h
ave, it's just a sample file name). Also note that comment lines</comment><comme
nt xml:space="preserve">must not come before a code block for the rst/st/epytext
 formats to work.</comment><paragraph>The file <literal>docstrings/doc1.dst.txt<
/literal> is a file filtered to a specific format
(typically plain text, reStructedText, or Epytext) from an original
&amp;quot;singleton&amp;quot; documentation file named <literal>docstrings/doc1.do.txt</
literal>. The <literal>.dst.txt</literal>
is the extension of a file filtered ready for being included in a doc
string (<literal>d</literal> for doc, <literal>st</literal> for string).</paragr
aph><paragraph>For making an Epydoc manual, the <literal>docstrings/doc1.do.txt<
/literal> file is
filtered to <literal>docstrings/doc1.epytext</literal> and renamed to
<literal>docstrings/doc1.dst.txt</literal>.  Then we run the preprocessor on the
<literal>basename.p.py</literal> file and create a real Python file
<literal>basename.py</literal>. Finally, we run Epydoc on this file. Alternative
ly, and
nowadays preferably, we use Sphinx for API documentation and then the
Doconce <literal>docstrings/doc1.do.txt</literal> file is filtered to
<literal>docstrings/doc1.rst</literal> and renamed to <literal>docstrings/doc1.d
st.txt</literal>. A
Sphinx directory must have been made with the right <literal>index.rst</literal>
 and
<literal>conf.py</literal> files. Going to this directory and typing <literal>ma
```

tutorial.xml

```
ke html</literal> makes
the HTML version of the Sphinx API documentation.</paragraph><paragraph>The next
 step is to produce the final pure Python source code. For
this purpose we filter <literal>docstrings/doc1.do.txt</literal> to plain text f
ormat
(<literal>docstrings/doc1.txt</literal>) and rename to <literal>docstrings/doc1.
dst.txt</literal>. The
preprocessor transforms the <literal>basename.p.py</literal> file to a standard
Python
file <literal>basename.py</literal>. The doc strings are now in plain text and w
ell
suited for Pydoc or reading by humans. All these steps are automated
by the <literal>insertdocstr.py</literal> script.  Here are the corresponding Un
ix
commands:</paragraph><literal_block xml:space="preserve"># make Epydoc API manua
l of basename module:
cd docstrings
doconce2format epytext doc1.do.txt
mv doc1.epytext doc1.dst.txt
cd ..
preprocess basename.p.py &gt; basename.py
epydoc basename

# make Sphinx API manual of basename module:
cd doc
doconce2format sphinx doc1.do.txt
mv doc1.rst doc1.dst.txt
cd ..
preprocess basename.p.py &gt; basename.py
cd docstrings/sphinx-rootdir  # sphinx directory for API source
make clean
make html
cd ../..

# make ordinary Python module files with doc strings:
cd docstrings
doconce2format plain doc1.do.txt
mv doc1.txt doc1.dst.txt
cd ..
preprocess basename.p.py &gt; basename.py

# can automate inserting doc strings in all .p.py files:
insertdocstr.py plain .
# (runs through all .do.txt files and filters them to plain format and
# renames to .dst.txt extension, then the script runs through all
# .p.py files and runs the preprocessor, which includes the .dst.txt
# files)</literal_block></section></section><section ids="warning-disclaimer" na
mes="warning/disclaimer"><title>Warning/Disclaimer</title><paragraph>Doconce can
 be viewed is a unified interface to a variety of
typesetting formats.  This interface is minimal in the sense that a
lot of typesetting features are not supported, for example, footnotes
and bibliography. For many documents the simple Doconce format is
sufficient, while in other cases you need more sophisticated
formats. Then you can just filter the Doconce text to a more
approprite format and continue working in this format only.  For
example, reStructuredText is a good alternative: it is more tagged
than Doconce and cannot be filtered to plain, untagged text, or wiki,
and the LaTeX output is not at all as clean, but it also has a lot
more typesetting and tagging features than Doconce.</paragraph></section></docum
```

”

tutorial.xml

”

```
ent>
```

”

”

”