

” **tutorial.do.txt** ”

TITLE: Doconce: Document Once, Include Everywhere  
 AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of Oslo  
 DATE: July 30, 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 everywhere?

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 everywhere". 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 comments, etc.).
- o Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

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

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

” **tutorial.do.txt** ”

- \* blocks of LaTeX mathematics can easily be included,
- \* there is support oforboth 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 =====

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".

Tables are also supported, e.g.,

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

```
!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:

## tutorial.do.txt

- 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"`.

Tables are also supported, e.g.,

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

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

Inline mathematics, such as `$_nu = \sin(x)$|$_v = sin(x)$`, 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, \backslash 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
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

”

**tutorial.do.txt**

”

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.

==== Seeing More of What Doconce Is ====

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder ``doc/tutorial.do.txt`` in the source code tree of Doconce. The Doconce example documentation displays both the source ``tutorial.do.txt`` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file ``lib/doconce/doc/doconce.do.txt`` in the Doconce source code tree. The same documentation appears in the doc string of the ``doconce`` module.

# 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 ``doconce.do.txt`` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the ``lib/doconce/doc`` directory which translates the more comprehensive documentation, ``doconce.do.txt``, to various formats. For example, to go from the LaTeX format to PDF, see ``latex.sh``.

==== 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.

”

”

”

” **tutorial.do.txt** ”

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, reStructuredText, 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:

```
!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
```

**tutorial.do.txt**

```
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
appropriate 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 Everywhere

Hans Petter Langtangen  
Simula Research Laboratory and University of Oslo

July 30, 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 everywhere?

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 everywhere". 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

## What Does Doconce Look Like?

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

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".

Tables are also supported, e.g.,



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


```

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

1. (for ordered) instead of the asterisk:

- (a) item 1
- (b) item 2
- (c) item 3

URLs with a link word are possible, as in [hpl](#). Just a file link goes like [tutorial.do.txt](#).

Tables are also supported, e.g.,



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

## Mathematics and Computer Code

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

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

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

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.

## Seeing More of What Doconce Is

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `doc/tutorial.do.txt` in the source code tree of Doconce. The Doconce example documentation displays both the source `tutorial.do.txt` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file `lib/doconce/doc/doconce.do.txt` in the Doconce source code tree. The same documentation appears in the doc string of the `doconce` module.

## 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:

**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
y
n
n
n
n
y
n
n
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
```

**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 `doconce.do.txt` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the `lib/doconce/doc` directory which translates the more comprehensive documentation, `doconce.do.txt`, to various formats. For example, to go from the LaTeX format to PDF, see `latex.sh`.

## 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, `reStructuredText`, 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 as 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 appropriate 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 Everywhere

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

**Date:** July 30, 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 everywhere”. 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructured-Text, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

## What Does Doconce Look Like?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the forming. 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 oforboth 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 ====

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"`.

Tables are also supported, e.g.,

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

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

1. (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](#).

Tables are also supported, e.g.,

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

## Mathematics and Computer Code

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

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

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

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.

## Seeing More of What Doconce Is

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `doc/tutorial.do.txt` in the source code tree of Doconce. The Doconce example documentation displays both the source `tutorial.do.txt` and the result of many other formats.



A more complete documentation of and motivation for Doconce appears in the file `lib/doconce/doc/doconce.do.txt` in the Doconce source code tree. The same documentation appears in the doc string of the doconce module.

## 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
y
n
n
n
n
y
```

```
n
n
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
```

## 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 `doconce.do.txt` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the `lib/doconce/doc` directory which translates the more comprehensive documentation, `doconce.do.txt`, to various formats. For example, to go from the LaTeX format to PDF, see `latex.sh`.

## 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, `reStructuredText`, 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 as 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 appropriate 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 18, 2010



# CONTENTS

<b>1</b>	<b>Doconce: Document Once, Include Everywhere</b>	<b>3</b>
1.1	The Doconce Concept . . . . .	3
1.2	What Does Doconce Look Like? . . . . .	3
1.3	A Subsection with Sample Text . . . . .	4
1.4	Mathematics and Computer Code . . . . .	5
1.5	Seeing More of What Doconce Is . . . . .	5
1.6	From Doconce to Other Formats . . . . .	6
1.7	The Doconce Documentation Strategy for User Manuals . . . . .	8
<b>2</b>	<b>Warning/Disclaimer</b>	<b>11</b>
<b>3</b>	<b>Indices and tables</b>	<b>13</b>





Contents:



# DOCONCE: DOCUMENT ONCE, INCLUDE EVERYWHERE

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

**Date** July 30, 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 everywhere”. 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

## 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 formatting. 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 =====
```

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"`.

Tables are also supported, e.g.,

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

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

1. (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`.

Tables are also supported, e.g.,

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

## 1.4 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 look 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 give you the right writing environment for plain text.

## 1.5 Seeing More of What Doconce Is

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `doc/tutorial.do.txt` in the source code tree of Doconce. The Doconce example documentation displays both the source `tutorial.do.txt` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file `lib/doconce/doc/doconce.do.txt` in the Doconce source code tree. The same documentation appears in the doc string of the `doconce` module.

## 1.6 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"`.

### 1.6.1 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.

### 1.6.2 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.

### 1.6.3 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
```

### 1.6.4 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.

### 1.6.5 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
Y
n
n
n
n
Y
n
n
```



```
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
```

## 1.6.6 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 `doconce.do.txt` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the `lib/doconce/doc` directory which translates the more comprehensive documentation, `doconce.do.txt`, to various formats. For example, to go from the LaTeX format to PDF, see `latex.sh`.

## 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, `reStructuredText`, 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 appropriate 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.



# INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

## tutorial.txt

TITLE: Doconce: Document Once, Include Everywhere

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

DATE: July 30, 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 everywhere?

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 everywhere". 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

### What Does Doconce Look Like?

-----

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

” **tutorial.txt** ”

- \* blocks of computer code can easily be included, also from source files,
- \* blocks of LaTeX mathematics can easily be included,
- \* there is support oforboth 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 =====

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"`.

Tables are also supported, e.g.,

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

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



” **tutorial.txt** ”

Lists can also have numbered items instead of bullets, just use an  
1. (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.

Tables are also supported, e.g.,

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

### Mathematics and Computer Code

-----

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

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

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

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
```

**tutorial.txt**

```
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.

### Seeing More of What Doconce Is

-----

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `doc/tutorial.do.txt` in the source code tree of Doconce. The Doconce example documentation displays both the source `tutorial.do.txt` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file `lib/doconce/doc/doconce.do.txt` in the Doconce source code tree. The same documentation appears in the doc string of the doconce module.

### 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`

” **tutorial.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
Y
n
n
n
n
Y
n
n
Y
Y
EOF

```

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

```

Unix/DOS> mv mydoc.rst sphinx-rootdir

```

” **tutorial.txt** ”

\*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
```

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 doconce.do.txt file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the lib/doconce/doc directory which translates the more comprehensive documentation, doconce.do.txt, to various formats. For example, to go from the LaTeX format to PDF, see latex.sh.

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.

” **tutorial.txt** ”

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, `reStructuredText`, 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
```

**tutorial.txt**

```
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 as 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 appropriate 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 Everywhere

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

DATE: July 30, 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 everywhere?

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 everywhere". 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

### What Does Doconce Look Like?

-----

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



## tutorial.epytext

- there is support oforboth 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 =====
```

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"`.

Tables are also supported, e.g.,

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

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 1. (for ordered) instead of the asterisk:

” **tutorial.epytext** ”

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>}`.

Tables are also supported, e.g.,

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

### Mathematics and Computer Code

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

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

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)
```

## tutorial.epytext

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.

### Seeing More of What Doconce Is

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `C{doc/tutorial.do.txt}` in the source code tree of Doconce. The Doconce example documentation displays both the source `C{tutorial.do.txt}` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file `C{lib/doconce/doc/doconce.do.txt}` in the Doconce source code tree. The same documentation appears in the doc string of the `C{doconce}` module.

### 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::

” **tutorial.epytext** ”

```
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::

” **tutorial.epytext** ”

```

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
Y
n
n
n
Y
n
n
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}

”

**tutorial.epytext**

”

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
```

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{doconce.do.txt}` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the `C{lib/doconce/doc}` directory which translates the more comprehensive documentation, `C{doconce.do.txt}`, to various formats. For example, to go from the LaTeX format to PDF, see `C{latex.sh}`.

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.

”

”

”

” **tutorial.epytext** ”

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, reStructuredText, 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
```

**tutorial.epytext**

```
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 as 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 appropriate 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.wiki

```
#summary Doconce: Document Once, Include Everywhere
<center><h3>Hans Petter Langtangen<br>Simula Research Laboratory and University
of Oslo</h3></center>
```

```
<center><h3>July 30, 2010</h3></center>
```

```
<!-- 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 everywhere?

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 and avoiding duplication of information. The slogan is:
"Write once, include everywhere". 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
comments, etc.).
```

```
# Doconce is a simple and minimally tagged markup language that can
be used for the above purpose. 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, reStructuredText, Sphinx, XML,
OpenOffice/Word, Epytext, PDF, XML - and even plain text (with
tags removed for clearer reading).
```

== What Does Doconce Look Like? ==

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the formatting. 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 oforboth LaTeX and text-like inline mathematics,
- \* figures with captions, URLs with links, labels and references are supported,

## tutorial.wiki

- \* 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 =====
```

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"`.

Tables are also supported, e.g.,

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

```
}}}
```

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 `#` (for ordered) instead of the asterisk:

- # item 1
- # item 2
- # item 3

URLs with a link word are possible, as in `[http://folk.uio.no/hpl hpl]`. Just a file link goes like `[tutorial.do.txt]`.

Tables are also supported, e.g.,

## tutorial.wiki

| 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

```
{\begin{code}
from math import sin, pi
def myfunc(x):
    return sin(pi*x)

import integrate
I = integrate.trapezoidal(myfunc, 0, pi, 100)
\end{code}}
```

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.

== Seeing More of What Doconce Is ==

After the quick syntax tour above, we recommend to read the Doconce

” **tutorial.wiki** ”

source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder 'doc/tutorial.do.txt' in the source code tree of Doconce. The Doconce example documentation displays both the source 'tutorial.do.txt' and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file 'lib/doconce/doc/doconce.do.txt' in the Doconce source code tree. The same documentation appears in the doc string of the 'doconce' module.

```
<!-- Example on including another Doconce file: -->
```

```
== 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
}}}
```

## tutorial.wiki

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
y
n
n
n
n
y
n

```

## tutorial.wiki

```

n
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
}}}
```

=== 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 'doconce.do.txt' file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the 'lib/doconce/doc' directory which translates the more comprehensive documentation, 'doconce.do.txt', to various formats. For example, to go from the LaTeX format to PDF, see 'latex.sh'.

== 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.

## tutorial.wiki

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"
}
}
}
<!-- -->
<!-- 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, reStructuredText, 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
```

**tutorial.wiki**

```
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 appropriate 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 18, 2010



# CONTENTS

|          |                                                               |           |
|----------|---------------------------------------------------------------|-----------|
| <b>1</b> | <b>Doconce: Document Once, Include Everywhere</b>             | <b>3</b>  |
| 1.1      | The Doconce Concept . . . . .                                 | 3         |
| 1.2      | What Does Doconce Look Like? . . . . .                        | 3         |
| 1.3      | A Subsection with Sample Text . . . . .                       | 4         |
| 1.4      | Mathematics and Computer Code . . . . .                       | 5         |
| 1.5      | Seeing More of What Doconce Is . . . . .                      | 5         |
| 1.6      | From Doconce to Other Formats . . . . .                       | 6         |
| 1.7      | The Doconce Documentation Strategy for User Manuals . . . . . | 8         |
| <b>2</b> | <b>Warning/Disclaimer</b>                                     | <b>11</b> |
| <b>3</b> | <b>Indices and tables</b>                                     | <b>13</b> |



Contents:



# DOCONCE: DOCUMENT ONCE, INCLUDE EVERYWHERE

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

**Date** July 30, 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 everywhere”. 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 comments, etc.).
2. Doconce is a simple and minimally tagged markup language that can be used for the above purpose. 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, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).

## 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 formatting. 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 =====
```

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"`.

Tables are also supported, e.g.,

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

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

1. (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`.

Tables are also supported, e.g.,

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

## 1.4 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 look 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 give you the right writing environment for plain text.

## 1.5 Seeing More of What Doconce Is

After the quick syntax tour above, we recommend to read the Doconce source of the current tutorial and compare it with what you see in a browser, a PDF document, in plain text, and so forth. The Doconce source is found in the folder `doc/tutorial.do.txt` in the source code tree of Doconce. The Doconce example documentation displays both the source `tutorial.do.txt` and the result of many other formats.

A more complete documentation of and motivation for Doconce appears in the file `lib/doconce/doc/doconce.do.txt` in the Doconce source code tree. The same documentation appears in the doc string of the `doconce` module.

## 1.6 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"`.

### 1.6.1 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.

### 1.6.2 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.

### 1.6.3 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
```

### 1.6.4 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.

### 1.6.5 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
Y
n
n
n
n
Y
n
n
```

```
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
```

### 1.6.6 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 `doconce.do.txt` file, gives a quick introduction to how Doconce is used in a real case.

There is another demo in the `lib/doconce/doc` directory which translates the more comprehensive documentation, `doconce.do.txt`, to various formats. For example, to go from the LaTeX format to PDF, see `latex.sh`.

## 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, `reStructuredText`, 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 appropriate 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.





# INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

## 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-everywhere" names="doconce:\ document\ once,\ include\ everywhere"><title>Doconce: Document Once, Include Everywhere</title><field_list><field><field_name>Author</field_name><field_body><paragraph>Hans Petter Langtangen, Simula Research Laboratory and University of Oslo</paragraph></field><field><field_name>Date</field_name><field_body><paragraph>July 30, 2010</paragraph></field></field_list><comment xml:space="preserve">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 everywhere?</comment><paragraph>If any of these questions are of interest, you should keep on reading.</paragraph><section ids="the-doconce-concept" names="the\ doconce\ concept"><title>The Doconce Concept</title><paragraph>Doconce is two things:</paragraph><block_quote><enumerated_list enumtype="arabic" prefix="" suffix="."><list_item><paragraph>Doconce is a working strategy for documenting software in a single place and avoiding duplication of information. The slogan is: "Write once, include everywhere". 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 comments, etc.).</paragraph></list_item><list_item><paragraph>Doconce is a simple and minimally tagged markup language that can be used for the above purpose. The Doconce format looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Wiki, LaTeX, reStructuredText, Sphinx, XML, OpenOffice/Word, Epytext, PDF, XML - and even plain text (with tags removed for clearer reading).</paragraph></list_item></enumerated_list></block_quote></section><section ids="what-does-doconce-look-like" names="what\ does\ doconce\ look\ like?"><title>What Does Doconce Look Like?</title><paragraph>Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the formatting. For example,</paragraph><block_quote><bullet_list bullet="*"><list_item><paragraph>bullet lists arise from lines starting with an asterisk,</paragraph></list_item><list_item><paragraph><emphasis>emphasized words</emphasis> are surrounded by asterisks,</paragraph></list_item><list_item><paragraph><strong>words in boldface</strong> are surrounded by underscores,</paragraph></list_item><list_item><paragraph>words from computer code are enclosed in back quotes and then typeset verbatim,</paragraph></list_item><list_item><paragraph>blocks of computer code can easily be included, also from source files,</paragraph></list_item><list_item><paragraph>blocks of LaTeX mathematics can easily be included,</paragraph></list_item><list_item><paragraph>there is support for both LaTeX and text-like inline mathematics,</paragraph></list_item><list_item><paragraph>figures with captions, URLs with links, labels and references are supported,</paragraph></list_item></bullet_list></block_quote></section></document>
```

## tutorial.xml

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 Doconce  
 e format:</paragraph><literal\_block xml:space="preserve">==== A Subsection with  
 Sample Text =====</literal\_block>

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:<quot;tutorial.do.txt".

Tables are also supported, e.g.,

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

</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</paragraph><enumera  
 ted\_list enumtype="arabic" prefix="" suffix="."><list\_item><paragraph>(for order  
 ed) instead of the asterisk:</paragraph></list\_item></enumerated\_list><block\_quo  
 te><enumerated\_list enumtype="arabic" prefix="" suffix="."><list\_item><paragraph  
 >item 1</paragraph></list\_item><list\_item><paragraph>item 2</paragraph></list\_it  
 em><list\_item><paragraph>item 3</paragraph></list\_item></enumerated\_list></block  
 \_quote><paragraph>URLs with a link word are possible, as in <reference name="hpl  
 " refuri="http://folk.uio.no/hpl">hpl</reference><target ids="hpl" names="hpl" r  
 efuri="http://folk.uio.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"/>.</paragraph><paragraph>Tables are also support  
 ed, e.g.,</paragraph><table><tgrouop cols="3"><colspec colwidth="12"/><colspec co  
 lwidth="12"/><colspec colwidth="12"/><thead><row><entry><paragraph>time</paragra  
 ph></entry><entry><paragraph>velocity</paragraph></entry><entry><paragraph>accel

## tutorial.xml

```

eration</paragraph></entry></row></thead><tbody><row><entry><paragraph>0.0</para
graph></entry><entry><paragraph>1.4186</paragraph></entry><entry><paragraph>-5.0
1</paragraph></entry></row><row><entry><paragraph>2.0</paragraph></entry><entry>
<paragraph>1.376512</paragraph></entry><entry><paragraph>11.919</paragraph></ent
ry></row><row><entry><paragraph>4.0</paragraph></entry><entry><paragraph>1.1E+1<
/paragraph></entry><entry><paragraph>14.717624</paragraph></entry></row></tbody>
</tgroup></table></section><section ids="mathematics-and-computer-code" names="m
athematics\ and\ computer\ code"><title>Mathematics and Computer Code</title><pa
ragraph>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} \&=\& \nabla^2 u + f,\label{myeq1}\\
{\partial v\over\partial t} \&=\& \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 "mynote.do.txt"</literal>
on a line starting with (another) hash sign. Doconce documents have
extension <literal>.do.txt</literal>. The <literal>do</literal> part stands for d
oconce, while the
trailing <literal>.txt</literal> denotes a text document so that editors gives y
ou the
right writing enviroment for plain text.</paragraph></section><section ids="seei
ng-more-of-what-doconce-is" names="seeing\ more\ of\ what\ doconce\ is"><title>S
eeing More of What Doconce Is</title><paragraph>After the quick syntax tour abov
e, we recommend to read the Doconce
source of the current tutorial and compare it with what you see in
a browser, a PDF document, in plain text, and so forth.
The Doconce source is found in the folder <literal>doc/tutorial.do.txt</literal>
in the
source code tree of Doconce. The Doconce example documentation
displays both the source <literal>tutorial.do.txt</literal> and the result of ma
ny other
formats.</paragraph><paragraph>A more complete documentation of and motivation f

```

## tutorial.xml

or Doconce appears in the file `<literal>lib/doconce/doc/doconce.do.txt</literal>` in the Doconce source code tree. The same documentation appears in the doc string of the `<literal>doconce</literal>` module.

Example on including another Doconce file:

From Doconce to Other Formats

Transformation of a Doconce document to various other formats applies the script `<literal>doconce2format</literal>`:

Unix/DOS> doconce2format format mydoc.do.txt

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. For example:

Unix/DOS> doconce2format LaTeX mydoc.do.txt -Dextra\_sections

The variable `<literal>FORMAT</literal>` is always defined as the current format when running `<literal>preprocess</literal>`. That is, in the last example, `<literal>FORMAT</literal>` is defined as `<literal>LaTeX</literal>`. Inside the Doconce document one can then perform format specific actions through tests like `<literal>#if FORMAT == "LaTeX"</literal>`.

HTML

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

Unix/DOS> doconce2format HTML mydoc.do.txt

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

LaTeX

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

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

**Step 1.** Filter the doconce text to a pre-LaTeX form `<literal>mydoc.p.tex</literal>` for

Unix/DOS> doconce2format LaTeX mydoc.do.txt

LaTeX-specific commands (`"newcommands"`) in math formulas and similar can be placed in a file `<literal>newcommands.tex</literal>`. If this file is present, it is included in the LaTeX document so that your commands are defined.

**Step 2.** Run `<literal>ptex2tex</literal>` (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</literal_block><paragraph>The <literal>
ptex2tex</literal> tool makes it possible to easily switch between many
different fancy formattings of computer or verbatim code in LaTeX
documents.
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>

```

| " | tutorial.xml                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | " |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|
|   | <pre> ; doconce2format plain mydoc.do.txt # results in mydoc.txt&lt;/literal_block&gt;&lt;/section&gt; &lt;section ids="restructuredtext" names="restructuredtext"&gt;&lt;title&gt;reStructure dText&lt;/title&gt;&lt;paragraph&gt;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 &lt;literal&gt;mydoc.rst&lt;/literal&gt;:&lt;/paragraph&gt;&lt;literal_block xml l:space="preserve"&gt;Unix/DOS&gt;&gt; doconce2format rst mydoc.do.txt&lt;/literal_block&gt;&lt; paragraph&gt;We may now produce various other formats:&lt;/paragraph&gt;&lt;literal_block xml l:space="preserve"&gt;Unix/DOS&gt;&gt; rst2html.py mydoc.rst &gt;&gt; mydoc.html # HTML Unix/DOS&gt;&gt; rst2latex.py mydoc.rst &gt;&gt; mydoc.tex # LaTeX Unix/DOS&gt;&gt; rst2xml.py mydoc.rst &gt;&gt; mydoc.xml # XML Unix/DOS&gt;&gt; rst2odt.py mydoc.rst &gt;&gt; mydoc.odt # OpenOffice&lt;/literal_block&gt; &lt;paragraph&gt;The OpenOffice file &lt;literal&gt;mydoc.odt&lt;/literal&gt; 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.&lt;/paragraph&gt;&lt;/section&gt; &lt;section ids="sphinx" names="sphinx"&gt;&lt;title&gt;Sphinx&lt;/title&gt;&lt;paragraph&gt;Sphinx docu ments can be created from a Doconce source in a few steps.&lt;/paragraph&gt;&lt;paragraph&gt; &lt;emphasis&gt;Step 1.&lt;/emphasis&gt; Translate Doconce into the Sphinx dialect of the reStructuredText format:&lt;/paragraph&gt;&lt;literal_block xml:space="preserve"&gt;Unix /DOS&gt;&gt; doconce2format sphinx mydoc.do.txt&lt;/literal_block&gt;&lt;paragraph&gt;&lt;emphasis&gt; Step 2.&lt;/emphasis&gt; Create a Sphinx root directory with a &lt;literal&gt;conf.py&lt;/liter al&gt; file, either manually or by using the interactive &lt;literal&gt;sphinx-quickstart&lt;/literal&gt; program. Here is a scripted version of the steps with the latter:&lt;/paragraph&gt;&lt;li teral_block xml:space="preserve"&gt;mkdir sphinx-rootdir sphinx-quickstart &amp;lt;&amp;lt;EOF sphinx-rootdir n - Name of My Sphinx Document Author version version .rst index Y n n n n Y n n Y Y EOF&lt;/literal_block&gt;&lt;paragraph&gt;&lt;emphasis&gt;Step 3.&lt;/emphasis&gt; Move the &lt;literal&gt;tut orial.rst&lt;/literal&gt; file to the Sphinx root directory:&lt;/paragraph&gt;&lt;literal_block xml:space="preserve"&gt;Unix/DOS&gt;&gt; mv mydoc.rst sphinx-rootdir&lt;/literal_block&gt;&lt;p aragraph&gt;&lt;emphasis&gt;Step 4.&lt;/emphasis&gt; Edit the generated &lt;literal&gt;index.rst&lt;/lit eral&gt; file so that &lt;literal&gt;mydoc.rst&lt;/literal&gt; is included, i.e., add &lt;literal&gt;mydoc&lt;/literal&gt; to the &lt;literal&gt;toctree&lt;/literal&gt; &gt; section so that it becomes:&lt;/paragraph&gt;&lt;literal_block xml:space="preserve"&gt;.. toctree::     :maxdepth: 2      mydoc&lt;/literal_block&gt;&lt;paragraph&gt;(The spaces before &lt;literal&gt;mydoc&lt;/literal&gt; a re important!)&lt;/paragraph&gt;&lt;paragraph&gt;&lt;emphasis&gt;Step 5.&lt;/emphasis&gt; Generate, for instance, an HTML version of the Sphinx source:&lt;/paragraph&gt;&lt;literal_block xml:sp </pre> |   |

” **tutorial.xml** ”

```

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> firefox _build/html/index.html</litera
l_block></section><section ids="demos" names="demos"><title>Demos</title><paragr
aph>The current text is generated from a Doconce format stored in the file:</par
agraph><literal_block xml:space="preserve">tutorial/tutorial.do.txt</literal_blo
ck><paragraph>The file <literal>make.sh</literal> in the <literal>tutorial</lite
ral> 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>doconce.do.txt</literal> fil
e,
gives a quick introduction to how Doconce is used in a real case.</paragraph><pa
ragraph>There is another demo in the <literal>lib/doconce/doc</literal> director
y which
translates the more comprehensive documentation, <literal>doconce.do.txt</litera
l>, to
various formats. For example, to go from the LaTeX format to PDF, see
<literal>latex.sh</literal>.</paragraph></section><section ids="the-do
conce-documentation-strategy-for-user-manuals" names="the\ doconce\ documentatio
n\ strategy\ for\ user\ manuals"><title>The Doconce Documentation Strategy for U
ser Manuals</title><paragraph>Doconce was particularly made for writing tutorial
s 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 &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, reStructuredText, or Epytext) from an original
&quot;singleton&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

```

**tutorial.xml**

```

<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
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

```



”

**tutorial.xml**

”

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 appropriate 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></document>