

” **tutorial.do.txt** ”

TITLE: Doconce: Document Once, Include Anywhere  
 AUTHOR: Hans Petter Langtangen at Simula Research Laboratory and University of Oslo  
 DATE: today

- \* When writing a note, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, Wiki, Word/OpenOffice, LaTeX, HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?
- \* Do you find it problematic that you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one place, and include it anywhere?

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

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

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- \* Doconce can convert to plain *\*untagged\** text, more desirable for computer programs and email.
- \* Doconce has less cluttered tagging of text.
- \* Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- \* Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.
- \* Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

” **tutorial.do.txt** ”

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

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

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

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- \* with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- \* with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

” **tutorial.do.txt** ”

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

- \* item 1
- \* item 2
- \* item 3

Lists can also have automatically numbered items instead of bullets,

- o item 1
- o item 2
- o item 3

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see `Chapter ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines  
!ec

The Doconce text above results in the following little document:

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

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

- \* item 1
- \* item 2
- \* item 3

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

- o item 1
- o item 2
- o item 3

” **tutorial.do.txt** ”

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

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

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other formats (but can still be useful for those who can read LaTeX syntax).

You can have blocks of computer code, starting and ending with `'!bc'` and `'!ec'` instructions, respectively. Such blocks look like

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

import integrate
```

” **tutorial.do.txt** ”

```
I = integrate.trapezoidal(myfunc, 0, pi, 100)
!ec
It is possible to add a specification of a (ptex2tex-style)
environment for typesetting the verbatim code block, e.g., '!bc xxx'
where 'xxx' is an identifier like 'pycod' for code snippet in Python,
'sys' for terminal session, etc. When Doconce is filtered to LaTeX,
these identifiers are used as in ptex2tex and defined in a
configuration file '.ptext2tex.cfg', while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):
```

```
!bc
# sphinx code-blocks: pycod=python cod=py cppcod=c++ sys=console
!ec
```

By default, 'pro' and 'cod' are 'python', 'sys' is 'console', while 'xpro' and 'xcod' are computer language specific for 'x' in 'f' (Fortran), 'c' (C), 'cpp' (C++), and 'py' (Python). # 'rb' (Ruby), 'pl' (Perl), and 'sh' (Unix shell).

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with '!bc pro', while a part of a file is copied into a '!bc cod' environment. What 'pro' and 'cod' mean is then defined through a '.ptex2tex.cfg' file for LaTeX and a 'sphinx code-blocks' comment for Sphinx.

Another document can be included by writing '#include "mynote.do.txt"' on a line starting with (another) hash sign. Doconce documents have extension 'do.txt'. The 'do' part stands for doconce, while the trailing '.txt' denotes a text document so that editors gives you the right writing enviroment for plain text.

```
==== Macros (Newcommands), Cross-References, Index, and Bibliography ====
label{newcommands}
```

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

” **tutorial.do.txt** ”

line (multi-line newcommands are too hard to parse with regular expressions).

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

# Example on including another Doconce file (using preprocess):

# #include "\_doconce2anything.do.txt"

===== Demos =====

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

docs/tutorial/tutorial.do.txt

!ec

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

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

===== Dependencies =====

If you make use of preprocessor directives in the Doconce source, either "Preprocess": "<http://code.google.com/p/preprocess>" or "Mako": "<http://www.makotemplates.org>" must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need "ptex2tex": "<http://code.google.com/p/ptex2tex>" and some style files that 'ptex2tex' potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires "docutils": "<http://docutils.sourceforge.net>". Making Sphinx documents requires of course "Sphinx": "<http://sphinx.pocoo.org>". All of the mentioned potential dependencies are pure Python packages which are easily installed.

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

Doconce can be viewed is a unified interface to a variety of

”

**tutorial.do.txt**

”

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 Anywhere

Hans Petter Langtangen<sup>1,2</sup>

<sup>1</sup>Simula Research Laboratory

<sup>2</sup>University of Oslo

Feb 20, 2011

- When writing a note, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, Wiki, Word/OpenOffice,  $\LaTeX$ , HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?
- Do you find it problematic that you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one place, and include it anywhere?

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

## 1 The Doconce Concept

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- Doconce can convert to plain *untagged* text, more desirable for computer programs and email.



- Doconce has less cluttered tagging of text.
- Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big)  $\text{\LaTeX}$  projects.
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki,  $\text{\LaTeX}$ , and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

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

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

## 2 What Does Doconce Look Like?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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  $\text{\LaTeX}$  mathematics can easily be included,
- there is support for both  $\text{\LaTeX}$  and text-like inline mathematics,
- figures with captions, URLs with links, labels and references are supported,
- comments can be inserted throughout the text,
- with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

    * item 1
    * item 2
    * item 3

Lists can also have automatically numbered items instead of bullets,

    o item 1
    o item 2
    o item 3

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

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

Doconce also allows inline comments such as [hpl: here I will make
some remarks to the text] for allowing authors to make notes. Inline
comments can be removed from the output by a command-line argument
(see Chapter ref{doconce2formats} for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines
```

The Doconce text above results in the following little document:

## 2.1 A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

URLs with a link word are possible, as in `hpl`. If the word is URL, the URL itself becomes the link name, as in `tutorial.do.txt`.

References to sections may use logical names as labels (e.g., a `"label"` command right after the section title), as in the reference to Chapter 2.1.

Doconce also allows inline comments such as `(hpl: here I will make some remarks to the text)` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter 3 for an example).

Tables are also supported, e.g.,

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

## 2.2 Mathematics and Computer Code

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

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

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)
```

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

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

By default, `pro` and `cod` are `python`, `sys` is `console`, while `xpro` and `xcod` are computer language specific for `x` in `f` (Fortran), `c` (C), `cpp` (C++), and `py` (Python).

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `bc pro!`, while a part of a file is copied into a `bc cod!` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for  $\LaTeX$  and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. Doconce documents have extension `do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you the right writing enviroment for plain text.

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

Doconce supports a type of macros via a  $\LaTeX$ -style *newcommand* construction. The newcommands defined in a file with name `newcommand_replace.tex`

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

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

### 3 From Doconce to Other Formats

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

---

Terminal

---

```
Unix/DOS> doconce format 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,

---

Terminal

---

```
Unix/DOS> doconce format 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"`.

Inline comments in the text are removed from the output by

---

Terminal

---

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

---

One can also remove such comments from the original Doconce file by running a helper script in the `bin` folder of the Doconce source code:

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

### 3.1 HTML

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

---

Terminal

---

```
Unix/DOS> doconce format HTML mydoc.do.txt
```

---

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

### 3.2 $\text{\LaTeX}$

Making an  $\text{\LaTeX}$  file `mydoc.tex` from `mydoc.do.txt` is done in two steps:

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

---

Terminal

---

```
Unix/DOS> doconce format LaTeX mydoc.do.txt
```

---

$\text{\LaTeX}$ -specific commands ("newcommands") in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see Section 2.3). If these files are present, they are included in the  $\text{\LaTeX}$  document so that your commands are defined.

**Step 2.** Run `ptex2tex` (if you have it) to make a standard  $\text{\LaTeX}$  file,

---

Terminal

---

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

---

or just perform a plain copy,

---

Terminal

---

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

---

Doconce generates a `.p.tex` file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font,

---

Terminal

---

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

---

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard  $\LaTeX$  "maketitle" heading is also available through

---

Terminal

---

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

---

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in  $\LaTeX$  documents. After any `bc sys!` command in the Doconce source you can insert verbatim block styles as defined in your `.ptex2tex.cfg` file, e.g., `bc sys cod!` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are over 30 styles to choose from.

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

---

Terminal

---

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvi2pdf mydoc
```

---

If one wishes to use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code, the `minted  $\LaTeX$`  package is needed. This package is included by running `doconce format` with the `-DMINTED` option:

---

Terminal

---

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

---

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

---

Terminal

---

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvi2pdf mydoc
```

---

The `-shell-escape` option is required because the `minted.sty` style file runs the `pygments` program to format code, and this program cannot be run from `latex` without the `-shell-escape` option.

### 3.3 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:

---

Terminal

---

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

---

### 3.4 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`:

---

Terminal

---

```
Unix/DOS> doconce format rst mydoc.do.txt
```

---

We may now produce various other formats:

---

Terminal

---

```
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.

### 3.5 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:

---

Terminal

---

```
Unix/DOS> doconce format 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:

---

Terminal

---

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

---

These statements are automated by the command

---

Terminal

---

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

---

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

---

Terminal

---

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

---

If you have figures in your document, the relative paths to those will be invalid when you work with `mydoc.rst` in the `sphinx-rootdir` directory. Either edit `mydoc.rst` so that figure file paths are correct, or simply copy your figure directory to `sphinx-rootdir` (if all figures are located in a subdirectory).

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

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

   mydoc
```

(The spaces before `mydoc` are important!)

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

---

Terminal

---

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

---

Many other formats are also possible.

**Step 6.** View the result:

---

Terminal

---

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

---

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

### 3.6 Google Code Wiki

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

---

Terminal

---

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

---

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

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

### 3.7 Tweaking the Doconce Output

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

### 3.8 Demos

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

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

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

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

### 3.9 Dependencies

If you make use of preprocessor directives in the Doconce source, either `Preprocess` or `Mako` must be installed. To make  $\text{\LaTeX}$  documents (without going through the `reStructuredText` format) you also need `ptex2tex` and some style files that `ptex2tex` potentially makes use of. Going from `reStructuredText` to formats such as XML, OpenOffice, HTML, and  $\text{\LaTeX}$  requires `docutils`. Making Sphinx documents requires of course Sphinx. All of the mentioned potential dependencies are pure Python packages which are easily installed.

## 4 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  $\text{\LaTeX}$  output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

# Doconce: Document Once, Include Anywhere

**Author:** Hans Petter Langtangen

**Date:** Feb 20, 2011

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

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

## The Doconce Concept

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- Doconce can convert to plain *untagged* text, more desirable for computer programs and email.
- Doconce has less cluttered tagging of text.
- Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.

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

Doconce was particularly written for the following sample applications:

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

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

## What Does Doconce Look Like?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,

- with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

- \* item 1
- \* item 2
- \* item 3

Lists can also have automatically numbered items instead of bullets,

- o item 1
- o item 2
- o item 3

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

## A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

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

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

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

Tables are also supported, e.g.,

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

## Mathematics and Computer Code

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

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

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

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, \; \text{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)
```

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

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

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. Doconce documents have extension `do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you the right writing enviroment for plain text.

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

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



Doconce. The definitions of newcommands in the `newcommands*.tex` files *must* appear on a single line (multi-line newcommands are too hard to parse with regular expressions).

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

## From Doconce to Other Formats

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

```
Unix/DOS> doconce format 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> doconce format 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"`.

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

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

One can also remove such comments from the original Doconce file by running a helper script in the `bin` folder of the Doconce source code:

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

## HTML

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

```
Unix/DOS> doconce format 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> doconce format LaTeX mydoc.do.txt
```

LaTeX-specific commands (“newcommands”) in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see the section [Macros \(Newcommands\), Cross-References, Index, and Bibliography](#)). If these files are present, they are included in the LaTeX document so that your commands are defined.

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

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

or just perform a plain copy:

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

Doconce generates a `.p.tex` file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font:

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX “maketitle” heading is also available through:

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formatings of computer or verbatim code in LaTeX documents. After any `!bc sys` command in the Doconce source you can insert verbatim block styles as defined in your `.ptex2tex.cfg` file, e.g., `!bc sys cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are over 30 styles to choose from.

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

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
```

If one wishes to use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code, the `minted` LaTeX package is needed. This package is included by running `doconce format` with the `-DMINTED` option:

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

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

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
```

The `-shell-escape` option is required because the `minted.sty` style file runs the `pygments` program to format code, and this program cannot be run from `latex` without the `-shell-escape` option.

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

## reStructuredText

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

```
Unix/DOS> doconce format 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> doconce format sphinx mydoc.do.txt
```

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

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

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

These statements are automated by the command:

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with `mydoc.rst` in the `sphinx-rootdir` directory. Either edit `mydoc.rst` so that figure file paths are correct, or simply copy your figure directory to `sphinx-rootdir` (if all figures are located in a subdirectory).

*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
```

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

## Google Code Wiki

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

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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

## Tweaking the Doconce Output

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

## Demos

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

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

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

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

## Dependencies

If you make use of preprocessor directives in the Doconce source, either [Preprocess](#) or [Mako](#) must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need [ptex2tex](#) and some style files that `ptex2tex` potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires [docutils](#). Making Sphinx documents requires of course [Sphinx](#). All of the mentioned potential dependencies are pure Python packages which are easily installed.

## 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 Anywhere Documentation**

***Release 1.0***

**Author**

February 20, 2011





# CONTENTS

<b>1</b>	<b>Doconce: Document Once, Include Anywhere</b>	<b>3</b>
<b>2</b>	<b>The Doconce Concept</b>	<b>5</b>
<b>3</b>	<b>What Does Doconce Look Like?</b>	<b>7</b>
3.1	A Subsection with Sample Text . . . . .	8
3.2	Mathematics and Computer Code . . . . .	9
3.3	Macros (Newcommands), Cross-References, Index, and Bibliography . . . . .	9
<b>4</b>	<b>From Doconce to Other Formats</b>	<b>11</b>
4.1	HTML . . . . .	11
4.2	LaTeX . . . . .	11
4.3	Plain ASCII Text . . . . .	13
4.4	reStructuredText . . . . .	13
4.5	Sphinx . . . . .	13
4.6	Google Code Wiki . . . . .	14
4.7	Tweaking the Doconce Output . . . . .	15
4.8	Demos . . . . .	15
4.9	Dependencies . . . . .	15
<b>5</b>	<b>Warning/Disclaimer</b>	<b>17</b>
<b>6</b>	<b>Indices and tables</b>	<b>19</b>



Contents:



# DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

**Author** Hans Petter Langtangen

**Date** Feb 20, 2011

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

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



# THE DOCONCE CONCEPT

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- Doconce can convert to plain *untagged* text, more desirable for computer programs and email.
- Doconce has less cluttered tagging of text.
- Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

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

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





# WHAT DOES DOCONCE LOOK LIKE?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

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

Lists can also have automatically numbered items instead of bullets,

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

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see `Chapter ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

### 3.1 A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

URLs with a link word are possible, as in `hpl`. If the word is URL, the URL itself becomes the link name, as in `tutorial.do.txt`.

References to sections may use logical names as labels (e.g., a "label" command right after the section title), as in the reference to the chapter *A Subsection with Sample Text*.

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

Tables are also supported, e.g.,

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

## 3.2 Mathematics and Computer Code

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

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

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

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

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

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other formats (but can still be useful for those who can read LaTeX syntax).

You can have blocks of computer code, starting and ending with `!bc` and `!ec` instructions, respectively. Such blocks look like

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

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

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

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

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptext2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. Doconce documents have extension `do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you the right writing enviroment for plain text.

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

Doconce supports a type of macros via a LaTeX-style *newcommand* construction. The newcommands defined in a file with name `newcommand_replace.tex` are expanded when Doconce is filtered to other formats, except for

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

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

# FROM DOCONCE TO OTHER FORMATS

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

```
Unix/DOS> doconce format 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> doconce format 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"`.

Inline comments in the text are removed from the output by

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

One can also remove such comments from the original Doconce file by running a helper script in the `bin` folder of the Doconce source code:

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

## 4.1 HTML

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

```
Unix/DOS> doconce format HTML mydoc.do.txt
```

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

## 4.2 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> doconce format LaTeX mydoc.do.txt
```

LaTeX-specific commands (“newcommands”) in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see the section *Macros (Newcommands), Cross-References, Index, and Bibliography*). If these files are present, they are included in the LaTeX document so that your commands are defined.

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

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

or just perform a plain copy,

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

Doconce generates a `.p.tex` file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font,

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX “maketitle” heading is also available through

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any `!bc sys` command in the Doconce source you can insert verbatim block styles as defined in your `.ptex2tex.cfg` file, e.g., `!bc sys cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are over 30 styles to choose from.

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

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
```

If one wishes to use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code, the `minted LaTeX` package is needed. This package is included by running `doconce format` with the `-DMINTED` option:

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

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

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
```

The `-shell-escape` option is required because the `minted.sty` style file runs the `pygments` program to format code, and this program cannot be run from `latex` without the `-shell-escape` option.

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

## 4.4 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> doconce format 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.

## 4.5 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> doconce format sphinx mydoc.do.txt
```

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

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

```
n
Y
Y
Y
EOF
```

These statements are automated by the command

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with `mydoc.rst` in the `sphinx-rootdir` directory. Either edit `mydoc.rst` so that figure file paths are correct, or simply copy your figure directory to `sphinx-rootdir` (if all figures are located in a subdirectory).

*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
```

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

## 4.6 Google Code Wiki

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

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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



## 4.7 Tweaking the Doconce Output

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

## 4.8 Demos

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

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

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

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

## 4.9 Dependencies

If you make use of preprocessor directives in the Doconce source, either [Preprocess](#) or [Mako](#) must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need [ptex2tex](#) and some style files that `ptex2tex` potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires [docutils](#). Making Sphinx documents requires of course [Sphinx](#). All of the mentioned potential dependencies are pure Python packages which are easily installed.



## **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

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

” **tutorial.txt** ”

Doconce: Document Once, Include Anywhere  
=====

Hans Petter Langtangen [1, 2]

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

Date: Feb 20, 2011

- \* When writing a note, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, Wiki, Word/OpenOffice, LaTeX, HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?
- \* Do you find it problematic that you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one place, and include it anywhere?

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

The Doconce Concept  
=====

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- \* Doconce can convert to plain *\*untagged\** text, more desirable for computer programs and email.
- \* Doconce has less cluttered tagging of text.
- \* Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.

” **tutorial.txt** ”

- \* Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.
- \* Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

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

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

#### What Does Doconce Look Like?

=====

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- \* with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined

## tutorial.txt

in or out of the text,

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

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

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

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

- \* item 1
- \* item 2
- \* item 3

Lists can also have automatically numbered items instead of bullets,

- o item 1
- o item 2
- o item 3

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

If the word is URL, the URL itself becomes the link name, as in `"URL":"tutorial.do.txt"`.

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see `Chapter ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

A Subsection with Sample Text

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



**tutorial.txt**

- \* item 1
- \* item 2
- \* item 3

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

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

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

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

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

Tables are also supported, e.g.,

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

### Mathematics and Computer Code

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

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

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::

## tutorial.txt

```

\begin{eqnarray}
\{\partial u\over\partial t\} \&=& \nabla^2 u + f, \text{label}\{myeq1\}\\
\{\partial v\over\partial t\} \&=& \nabla\cdot(q(u)\nabla v) + g
\end{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  cod
from math import sin, pi
def myfunc(x):
    return sin(pi*x)

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

```

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

```

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

```

By default, pro and cod are python, sys is console, while xpro and xcod are computer language specific for x in f (Fortran), c (C), cpp (C++), and py (Python).

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with !bc pro, while a part of a file is copied into a !bc cod environment. What pro and cod mean is then defined through a .ptex2tex.cfg file for LaTeX and a sphinx code-blocks comment for Sphinx.

Another document can be included by writing #include "mynote.do.txt" on a line starting with (another) hash sign. Doconce documents have extension do.txt. The do part stands for doconce, while the trailing .txt denotes a text document so that editors gives you the right writing enviroment for plain text.

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

---

Doconce supports a type of macros via a LaTeX-style \*newcommand\*

” **tutorial.txt** ”

construction. The newcommands defined in a file with name `newcommand_replace.tex` are expanded when Doconce is filtered to other formats, except for LaTeX (since LaTeX performs the expansion itself). Newcommands in files with names `newcommands.tex` and `newcommands_keep.tex` are kept unaltered when Doconce text is filtered to other formats, except for the Sphinx format. Since Sphinx understands LaTeX math, but not newcommands if the Sphinx output is HTML, it makes most sense to expand all newcommands. Normally, a user will put all newcommands that appear in math blocks surrounded by::

least if they contribute to make the raw LaTeX math text easier to read in the formats that cannot render LaTeX. Newcommands used elsewhere throughout the text will usually be placed in `newcommands_replace.tex` and expanded by Doconce. The definitions of newcommands in the `newcommands*.tex` files *must* appear on a single line (multi-line newcommands are too hard to parse with regular expressions).

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

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

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

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

```
Unix/DOS> doconce format 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> doconce format 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"`.

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

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

” **tutorial.txt** ”

One can also remove such comments from the original Doconce file by running a helper script in the bin folder of the Doconce source code::

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

HTML  
----

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

```
Unix/DOS> doconce format 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> doconce format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in files newcommands.tex, newcommands\_keep.tex, or newcommands\_replace.tex (see the section "Macros (Newcommands), Cross-References, Index, and Bibliography"). If these files are present, they are included in the LaTeX document so that your commands are defined.

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

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

or just perform a plain copy::

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

Doconce generates a .p.tex file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font::

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having

” **tutorial.txt** ”

institutions in common. The standard LaTeX "maketitle" heading is also available through::

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The ptex2tex tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any !bc sys command in the Doconce source you can insert verbatim block styles as defined in your .ptex2tex.cfg file, e.g., !bc sys cod for a code snippet, where cod is set to a certain environment in .ptex2tex.cfg (e.g., CodeIntended). There are over 30 styles to choose from.

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

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc      # if index
Unix/DOS> bibitem mydoc       # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
```

If one wishes to use the Minted\_Python, Minted\_Cpp, etc., environments in ptex2tex for typesetting code, the minted LaTeX package is needed. This package is included by running doconce format with the -DMINTED option::

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

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

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc      # if index
Unix/DOS> bibitem mydoc       # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
```

The -shell-escape option is required because the minted.sty style file runs the pygments program to format code, and this program cannot be run from latex without the -shell-escape option.

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

” **tutorial.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> doconce format 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> doconce format sphinx mydoc.do.txt
```

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

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

” **tutorial.txt** ”

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

These statements are automated by the command::

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with mydoc.rst in the sphinx-rootdir directory. Either edit mydoc.rst so that figure file paths are correct, or simply copy your figure directory to sphinx-rootdir (if all figures are located in a subdirectory).

\*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
```

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

Google Code Wiki

-----

”

**tutorial.txt**

”

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

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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

#### Tweaking the Doconce Output

-----

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

#### Demos

-----

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

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

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

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

#### Dependencies

-----



”

**tutorial.txt**

”

If you make use of preprocessor directives in the Doconce source, either Preprocess (<http://code.google.com/p/preprocess>) or Mako (<http://www.makotemplates.org>) must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need ptex2tex (<http://code.google.com/p/ptex2tex>) and some style files that ptex2tex potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires docutils (<http://docutils.sourceforge.net>). Making Sphinx documents requires of course Sphinx (<http://sphinx.pocoo.org>). All of the mentioned potential dependencies are pure Python packages which are easily installed.

**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 Anywhere  
 BY: Hans Petter Langtangen (Simula Research Laboratory, and University of Oslo)  
 ATE: today

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

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

### The Doconce Concept

=====

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- Doconce can convert to plain I{untagged} text, more desirable for computer programs and email.
- Doconce has less cluttered tagging of text.
- Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

- Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other

## tutorial.epytext

contexts in other formats, including plain HTML, Sphinx, or MS Word.

- Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at googlecode.com, and as LaTeX integrated in, e.g., a thesis.
- Quick memos, which start as plain text in email, then some small amount of Doconce tagging is added, before the memos can appear as MS Word documents or in wikis.

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

### What Does Doconce Look Like?

=====

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- 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,
- with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

- \* item 1
- \* item 2
- \* item 3

Lists can also have automatically numbered items instead of bullets,

- o item 1
- o item 2
- o item 3

## tutorial.epytext

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

If the word is URL, the URL itself becomes the link name, as in `"URL":"tutorial.do.txt"`.

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

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

References to sections may use logical names as labels (e.g., a `"label"` command right after the section title), as in the reference to the chapter "A Subsection with Sample Text".

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

” **tutorial.epytext** ”

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)
```

It is possible to add a specification of a (ptex2tex-style) environment for typesetting the verbatim code block, e.g., `C{!bc xxx}` where `C{xxx}` is an identifier like `C{pycod}` for code snippet in Python, `C{sys}` for terminal session, etc. When Doconce is filtered to LaTeX,

## tutorial.epytext

these identifiers are used as in ptex2tex and defined in a configuration file `C{.ptext2tex.cfg}`, while when filtering to Sphinx, one can have a comment line in the Doconce file for mapping the identifiers to legal language names for Sphinx (which equals the legal language names for Pygments)::

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

By default, `C{pro}` and `C{cod}` are `C{python}`, `C{sys}` is `C{console}`, while `C{xpro}` and `C{xcod}` are computer language specific for `C{x}` in `C{f}` (Fortran), `C{c}` (C), `C{cpp}` (C++), and `C{py}` (Python).

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `C{!bc pro}`, while a part of a file is copied into a `C{!bc cod}` environment. What `C{pro}` and `C{cod}` mean is then defined through a `C{.ptex2tex.cfg}` file for LaTeX and a `C{sphinx code-blocks}` comment for Sphinx.

Another document can be included by writing `C{#include "mynote.do.txt"}` on a line starting with (another) hash sign. Doconce documents have extension `C{do.txt}`. The `C{do}` part stands for doconce, while the trailing `C{.txt}` denotes a text document so that editors gives you the right writing enviroment for plain text.

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

-----

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

Recent versions of Doconce also offer cross referencing, typically one can define labels below (sub)sections, in figure captions, or in equations, and then refer to these later. Entries in an index can be defined and result in an index at the end for the LaTeX and Sphinx formats. Citations to literature, with an accompanying bibliography in a file, are also supported. The syntax of labels, references, citations, and the bibliography closely resembles that of LaTeX,

”

**tutorial.epytext**

”

making it easy for Doconce documents to be integrated in LaTeX projects (manuals, books). For further details on functionality and syntax we refer to the C{docs/manual/manual.do.txt} file (see the U{demo page<<https://doconce.googlecode.com/hg/trunk/docs/demos/manual/index.html>>} for various formats of this document).

### From Doconce to Other Formats

=====

Transformation of a Doconce document to various other formats applies the script C{doconce format}::

```
Unix/DOS> doconce format 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> doconce format 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"}.

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

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

One can also remove such comments from the original Doconce file by running a helper script in the C{bin} folder of the Doconce source code::

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

### HTML

----

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

```
Unix/DOS> doconce format HTML mydoc.do.txt
```

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

### LaTeX

-----

" **tutorial.epytext** "

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

I{Step 1.} Filter the doonce text to a pre-LaTeX form C{mydoc.p.tex} for C{ptex2tex}::

```
Unix/DOS> doonce format LaTeX mydoc.do.txt
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in files C{newcommands.tex}, C{newcommands\_keep.tex}, or C{newcommands\_replace.tex} (see the section "Macros (Newcommands), Cross-References, Index, and Bibliography").

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

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
```

Doonce generates a C{.p.tex} file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font::

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX "maketitle" heading is also available through::

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The C{ptex2tex} tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any C{!bc sys} command in the Doonce source you can insert verbatim block styles as defined in your C{.ptex2tex.cfg} file, e.g., C{!bc sys cod} for a code snippet, where C{cod} is set to a certain environment in C{.ptex2tex.cfg} (e.g., C{CodeIntended}). There are over 30 styles to choose from.

I{Step 3.} Compile C{mydoc.tex} and create the PDF file::

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc      # if index
Unix/DOS> bibitem mydoc       # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
```



## tutorial.epytext

If one wishes to use the `C{Minted_Python}`, `C{Minted_Cpp}`, etc., environments in `C{ptex2tex}` for typesetting code, the `C{minted}` LaTeX package is needed. This package is included by running `C{doconce format}` with the `C{-DMINTED}` option::

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

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

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc      # if index
Unix/DOS> bibitem mydoc       # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
```

The `C{-shell-escape}` option is required because the `C{minted.sty}` style file runs the `C{pygments}` program to format code, and this program cannot be run from `C{latex}` without the `C{-shell-escape}` option.

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

### reStructuredText

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

```
Unix/DOS> doconce format rst mydoc.do.txt
```

We may now produce various other formats::

```
Unix/DOS> rst2html.py mydoc.rst > mydoc.html # HTML
Unix/DOS> rst2latex.py mydoc.rst > mydoc.tex  # LaTeX
Unix/DOS> rst2xml.py mydoc.rst > mydoc.xml    # XML
Unix/DOS> rst2odt.py mydoc.rst > mydoc.odt    # OpenOffice
```

The OpenOffice file `C{mydoc.odt}` can be loaded into OpenOffice and saved in, among other things, the RTF format or the Microsoft Word format. That is, one can easily go from Doconce to Microsoft Word.

### Sphinx

”

**tutorial.epytext**

”

-----

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

I{Step 2.} Create a Sphinx root directory with a C{conf.py} file, either manually or by using the interactive C{sphinx-quickstart} program. Here is a scripted version of the steps with the latter::

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

These statements are automated by the command::

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with C{mydoc.rst} in the C{sphinx-rootdir} directory. Either edit C{mydoc.rst} so that figure file paths are correct, or simply copy your figure directory to C{sphinx-rootdir} (if all figures are located in a subdirectory).

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
```

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

## Google Code Wiki

-----

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

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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

## Tweaking the Doconce Output

-----

Occasionally, one would like to tweak the output in a certain format from Doconce. One example is figure filenames when transforming Doconce to reStructuredText. Since Doconce does not know if the

”

”

”

”

**tutorial.epytext**

”

C{.rst} file is going to be filtered to LaTeX or HTML, it cannot know if C{.eps} or C{.png} is the most appropriate image filename. The solution is to use a text substitution command or code with, e.g., sed, perl, python, or scitools subst, to automatically edit the output file from Doconce. It is then wise to run Doconce and the editing commands from a script to automate all steps in going from Doconce to the final format(s). The C{make.sh} files in C{docs/manual} and C{docs/tutorial} constitute comprehensive examples on how such scripts can be made.

**Demos**

-----

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

docs/tutorial/tutorial.do.txt

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

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

**Dependencies**

-----

If you make use of preprocessor directives in the Doconce source, either U{Preprocess<<http://code.google.com/p/preprocess>>} or U{Mako<<http://www.makotemplates.org>>} must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need U{ptex2tex<<http://code.google.com/p/ptex2tex>>} and some style files that C{ptex2tex} potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires U{docutils<<http://docutils.sourceforge.net>>}. Making Sphinx documents requires of course U{Sphinx<<http://sphinx.pocoo.org>>}. All of the mentioned potential dependencies are pure Python packages which are easily installed.

**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,

”

”

”

”

## tutorial.epytext

”

and the LaTeX output is not at all as clean, but it also has a lot more typesetting and tagging features than Doconce.

## tutorial.gwiki

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

==== Feb 20, 2011 ====

\* When writing a note, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, Wiki, Word/OpenOffice, LaTeX, HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?

\* Do you find it problematic that you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one place, and include it anywhere?

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

== The Doconce Concept ==

Doconce is two things:

# Doconce is a very simple and minimally tagged markup language that look like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

\* Doconce can convert to plain \*untagged\* text, more desirable for computer programs and email.

\* Doconce has less cluttered tagging of text.

\* Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.

\* Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.

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

Doconce was particularly written for the following sample applications:

\* Large books written in LaTeX, but where many pieces (computer demos, projects, examples) can be written in Doconce to appear in other contexts in other

## tutorial.gwiki

er formats, including plain HTML, Sphinx, or MS Word.

- \* Software documentation, primarily Python doc strings, which one wants to appear as plain untagged text for viewing in Pydoc, as reStructuredText for use with Sphinx, as wiki text when publishing the software at [googlecode.com](http://googlecode.com), and as LaTeX integrated in, e.g., a thesis.

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

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

== What Does Doconce Look Like? ==

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- \* with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- \* with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

- \* item 1
- \* item 2
- \* item 3

Lists can also have automatically numbered items instead of bullets,

- o item 1
- o item 2
- o item 3

## tutorial.gwiki

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines  
}}}

The Doconce text above results in the following little document:

==== A Subsection with Sample Text ====

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

- \* item 1
- \* item 2
- \* item 3

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

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

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

References to sections may use logical names as labels (e.g., a "label" command right after the section title), as in the reference to the chapter `[#A_Subsection_with_Sample_Text]`.

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



## tutorial.gwiki

Tables are also supported, e.g.,

	<i>*time*</i>	<i>*velocity*</i>	<i>*acceleration*</i>
	0.0	1.4186	-5.01
	2.0	1.376512	11.919
	4.0	1.1E+1	14.717624

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

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

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

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

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, \text{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}}
```

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

```
{\begin{code}
# sphinx code-blocks: pycod=python cod=py cppcod=c++ sys=console
\end{code}}
```

By default, `'pro'` and `'cod'` are `'python'`, `'sys'` is `'console'`,

## tutorial.gwiki

while 'xpro' and 'xcod' are computer language specific for 'x' in 'f' (Fortran), 'c' (C), 'cpp' (C++), and 'py' (Python).  
 <wiki:comment> 'rb' (Ruby), 'pl' (Perl), and 'sh' (Unix shell). </wiki:comment>

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with '!bc pro', while a part of a file is copied into a '!bc cod' environment. What 'pro' and 'cod' mean is then defined through a '.ptex2tex.cfg' file for LaTeX and a 'sphinx code-blocks' comment for Sphinx.

Another document can be included by writing '#include "mynote.do.txt"' on a line starting with (another) hash sign. Doconce documents have extension 'do.txt'. The 'do' part stands for doconce, while the trailing '.txt' denotes a text document so that editors gives you the right writing environment for plain text.

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

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

Recent versions of Doconce also offer cross referencing, typically one can define labels below (sub)sections, in figure captions, or in equations, and then refer to these later. Entries in an index can be defined and result in an index at the end for the LaTeX and Sphinx formats. Citations to literature, with an accompanying bibliography in a file, are also supported. The syntax of labels, references, citations, and the bibliography closely resembles that of LaTeX, making it easy for Doconce documents to be integrated in LaTeX projects (manuals, books). For further details on functionality and syntax we refer to the 'docs/manual/manual.do.txt' file (see the <https://doconce.googlecode.com/hg/trunk/docs/demos/manual/index.html> demo

## tutorial.gwiki

page] for various formats of this document).

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

== From Doconce to Other Formats ==

Transformation of a Doconce document to various other formats applies the script 'doconce format':

```
{
Unix/DOS> doconce format 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> doconce format 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"'.

Inline comments in the text are removed from the output by

```
{
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
}
```

One can also remove such comments from the original Doconce file by running a helper script in the 'bin' folder of the Doconce source code:

```
{
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
}
```

This action is convenient when a Doconce document reaches its final form.

==== HTML ====

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

```
{
Unix/DOS> doconce format HTML mydoc.do.txt
}
```

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

==== LaTeX ====

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

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

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

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

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

```
{
Unix/DOS> doconce format LaTeX mydoc.do.txt
```

## tutorial.gwiki

```
}}}
```

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in files 'newcommands.tex', 'newcommands\_keep.tex', or 'newcommands\_replace.tex' (see the section [Macros\_(Newcommands),\_Cross-References,\_Index,\_and\_Bibliography]).

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

\*Step 2.\* Run 'ptex2tex' (if you have it) to make a standard LaTeX file,

```
{
{
{
Unix/DOS> ptex2tex mydoc
}
}
}
```

or just perform a plain copy,

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

Doconce generates a '.p.tex' file with some preprocessor macros.

For example, to enable font Helvetica instead of the standard Computer Modern font,

```
{
{
{
Unix/DOS> ptex2tex -DHELVETICA mydoc
}
}
}
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX "maketitle" heading is also available through

```
{
{
{
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
}
}
}
```

The 'ptex2tex' tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any '!bc sys' command in the Doconce source you can insert verbatim block styles as defined in your '.ptex2tex.cfg' file, e.g., '!bc sys cod' for a code snippet, where 'cod' is set to a certain environment in '.ptex2tex.cfg' (e.g., 'CodeIntended'). There are over 30 styles to choose from.

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

```
{
{
{
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
}
}
}
```

If one wishes to use the 'Minted\_Python', 'Minted\_Cpp', etc., environments in 'ptex2tex' for typesetting code, the 'minted' LaTeX package is needed. This package is included by running 'doconce format' with the '-DMINTED' option:

```
{
{
{
Unix/DOS> ptex2tex -DMINTED mydoc
}
}
}
```

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

```
{
{
{
Unix/DOS> latex -shell-escape mydoc
}
}
}
```

” **tutorial.gwiki** ”

```

Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc      # if index
Unix/DOS> bibitem mydoc       # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
}}}
```

The `'-shell-escape'` option is required because the `'minted.sty'` style file runs the `'pygments'` program to format code, and this program cannot be run from `'latex'` without the `'-shell-escape'` option.

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

==== reStructuredText ====

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

```

{{{
Unix/DOS> doconce format 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> doconce format 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

## tutorial.gwiki

```
version
version
.rst
index
n
Y
n
n
n
n
Y
n
n
Y
Y
Y
EOF
}}}
```

```
These statements are automated by the command
{{{
Unix/DOS> doconce sphinx_dir mydoc.do.txt
}}}
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with 'mydoc.rst' in the 'sphinx-rootdir' directory. Either edit 'mydoc.rst' so that figure file paths are correct, or simply copy your figure directory to 'sphinx-rootdir' (if all figures are located in a subdirectory).

\*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
}}}
```

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

## tutorial.gwiki

```
<wiki:comment> Desired extension: sphinx can utilize a "pycod" or "c++cod" </wiki:comment>
<wiki:comment> instruction as currently done in latex for ptex2tex and write </wiki:comment>
<wiki:comment> out the right code block name accordingly. </wiki:comment>
```

==== Google Code Wiki ====

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

```
{{{
Unix/DOS> doconce format gwiki mydoc.do.txt
}}}
```

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

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

==== Tweaking the Doconce Output ====

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

==== Demos ====

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

```
{{{
docs/tutorial/tutorial.do.txt
}}}
```

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

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

”

**tutorial.gwiki**

”

==== Dependencies ====

If you make use of preprocessor directives in the Doconce source, either [<http://code.google.com/p/preprocess> Preprocess] or [<http://www.makotemplates.org> Makol] must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need [<http://code.google.com/p/ptex2tex> ptex2tex] and some style files that 'ptex2tex' potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires [<http://docutils.sourceforge.net> docutils]. Making Sphinx documents requires of course [<http://sphinx.pocoo.org> Sphinx]. All of the mentioned potential dependencies are pure Python packages which are easily installed.

== 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 Anywhere Documentation**

***Release 1.0***

**Author**

February 20, 2011



# CONTENTS

<b>1</b>	<b>Doconce: Document Once, Include Anywhere</b>	<b>3</b>
<b>2</b>	<b>The Doconce Concept</b>	<b>5</b>
<b>3</b>	<b>What Does Doconce Look Like?</b>	<b>7</b>
3.1	A Subsection with Sample Text . . . . .	8
3.2	Mathematics and Computer Code . . . . .	9
3.3	Macros (Newcommands), Cross-References, Index, and Bibliography . . . . .	9
<b>4</b>	<b>From Doconce to Other Formats</b>	<b>11</b>
4.1	HTML . . . . .	11
4.2	LaTeX . . . . .	11
4.3	Plain ASCII Text . . . . .	13
4.4	reStructuredText . . . . .	13
4.5	Sphinx . . . . .	13
4.6	Google Code Wiki . . . . .	14
4.7	Tweaking the Doconce Output . . . . .	15
4.8	Demos . . . . .	15
4.9	Dependencies . . . . .	15
<b>5</b>	<b>Warning/Disclaimer</b>	<b>17</b>
<b>6</b>	<b>Indices and tables</b>	<b>19</b>



Contents:



# DOCONCE: DOCUMENT ONCE, INCLUDE ANYWHERE

**Author** Hans Petter Langtangen

**Date** Feb 20, 2011

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

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





# THE DOCONCE CONCEPT

Doconce is two things:

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

A wide range of markup languages exist. For example, reStructuredText and Sphinx have recently become popular. So why another one?

- Doconce can convert to plain *untagged* text, more desirable for computer programs and email.
- Doconce has less cluttered tagging of text.
- Doconce has better support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has stronger support for mathematical typesetting, and has many features for being integrated with (big) LaTeX projects.
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or Wiki document.

Doconce was particularly written for the following sample applications:

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

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



# WHAT DOES DOCONCE LOOK LIKE?

Doconce text looks like ordinary text, but there are some almost invisible text constructions that allow you to control the 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,
- with a simple preprocessor, which is integrated, one can include other documents (files) and large portions of text can be defined in or out of the text,
- with the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

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

Lists can also have automatically numbered items instead of bullets,

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

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

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see `Chapter ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

### 3.1 A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

URLs with a link word are possible, as in `hpl`. If the word is URL, the URL itself becomes the link name, as in `tutorial.do.txt`.

References to sections may use logical names as labels (e.g., a "label" command right after the section title), as in the reference to the chapter *A Subsection with Sample Text*.

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

Tables are also supported, e.g.,

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

## 3.2 Mathematics and Computer Code

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

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

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

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

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

Of course, such blocks only looks nice in LaTeX. The raw LaTeX syntax appears in all other formats (but can still be useful for those who can read LaTeX syntax).

You can have blocks of computer code, starting and ending with `!bc` and `!ec` instructions, respectively. Such blocks look like

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

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

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

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

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptext2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. Doconce documents have extension `do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you the right writing enviroment for plain text.

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

Doconce supports a type of macros via a LaTeX-style *newcommand* construction. The newcommands defined in a file with name `newcommand_replace.tex` are expanded when Doconce is filtered to other formats, except for

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

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

# FROM DOCONCE TO OTHER FORMATS

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

```
Unix/DOS> doconce format 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> doconce format 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"`.

Inline comments in the text are removed from the output by

```
Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments
```

One can also remove such comments from the original Doconce file by running a helper script in the `bin` folder of the Doconce source code:

```
Unix/DOS> doconce remove_inline_comments mydoc.do.txt
```

This action is convenient when a Doconce document reaches its final form.

## 4.1 HTML

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

```
Unix/DOS> doconce format HTML mydoc.do.txt
```

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

## 4.2 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> doconce format LaTeX mydoc.do.txt
```

LaTeX-specific commands (“newcommands”) in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see the section *Macros (Newcommands), Cross-References, Index, and Bibliography*). If these files are present, they are included in the LaTeX document so that your commands are defined.

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

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

or just perform a plain copy,

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

Doconce generates a `.p.tex` file with some preprocessor macros. For example, to enable font Helvetica instead of the standard Computer Modern font,

```
Unix/DOS> ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX “maketitle” heading is also available through

```
Unix/DOS> ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any `!bc sys` command in the Doconce source you can insert verbatim block styles as defined in your `.ptex2tex.cfg` file, e.g., `!bc sys cod` for a code snippet, where `cod` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeIntended`). There are over 30 styles to choose from.

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

```
Unix/DOS> latex mydoc
Unix/DOS> latex mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex mydoc
Unix/DOS> dvipdf mydoc
```

If one wishes to use the `Minted_Python`, `Minted_Cpp`, etc., environments in `ptex2tex` for typesetting code, the `minted LaTeX` package is needed. This package is included by running `doconce format` with the `-DMINTED` option:

```
Unix/DOS> ptex2tex -DMINTED mydoc
```

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

```
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> makeindex mydoc    # if index
Unix/DOS> bibitem mydoc      # if bibliography
Unix/DOS> latex -shell-escape mydoc
Unix/DOS> dvipdf mydoc
```

The `-shell-escape` option is required because the `minted.sty` style file runs the `pygments` program to format code, and this program cannot be run from `latex` without the `-shell-escape` option.



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

## 4.4 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> doconce format 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.

## 4.5 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> doconce format sphinx mydoc.do.txt
```

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

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

```
n
Y
Y
Y
EOF
```

These statements are automated by the command

```
Unix/DOS> doconce sphinx_dir mydoc.do.txt
```

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

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

If you have figures in your document, the relative paths to those will be invalid when you work with `mydoc.rst` in the `sphinx-rootdir` directory. Either edit `mydoc.rst` so that figure file paths are correct, or simply copy your figure directory to `sphinx-rootdir` (if all figures are located in a subdirectory).

*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
```

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

## 4.6 Google Code Wiki

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

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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

## 4.7 Tweaking the Doconce Output

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

## 4.8 Demos

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

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

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

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

## 4.9 Dependencies

If you make use of preprocessor directives in the Doconce source, either [Preprocess](#) or [Mako](#) must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need [ptex2tex](#) and some style files that `ptex2tex` potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires [docutils](#). Making Sphinx documents requires of course [Sphinx](#). All of the mentioned potential dependencies are pure Python packages which are easily installed.



## **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

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

"	tutorial.xml	"
	<pre> &lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;!DOCTYPE document PUBLIC "-//IDN docutils.sourceforge.net//DTD Docutils Generic //EN//XML" "http://docutils.sourceforge.net/docs/ref/docutils.dtd"&gt; &lt;!-- Generated by Docutils 0.8 --&gt; &lt;document source="tutorial.rst"&gt;&lt;comment xml:space="preserve"&gt;Automatically gene rated reST file from Doconce source (http://code.google.com/p/doconce/)&lt;/comment&gt;&lt;section ids="doconce-document-once -include-anywhere" names="doconce:\ document\ once,\ include\ anywhere"&gt;&lt;title&gt;D oconce: Document Once, Include Anywhere&lt;/title&gt;&lt;field_list&gt;&lt;field&gt;&lt;field_name&gt;Au thor&lt;/field_name&gt;&lt;field_body&gt;&lt;paragraph&gt;Hans Petter Langtangen&lt;/paragraph&gt;&lt;/fiel d_body&gt;&lt;/field&gt;&lt;field&gt;&lt;field_name&gt;Date&lt;/field_name&gt;&lt;field_body&gt;&lt;paragraph&gt;Feb 20 , 2011&lt;/paragraph&gt;&lt;bullet_list bullet="*"&gt;&lt;list_item&gt;&lt;paragraph&gt;When writing a n ote, report, manual, etc., do you find it difficult to choose the typesetting format? That is, to choose between plain (email-like) text, Wiki, Word/OpenOffice, LaTeX, HTML, reStructuredText, Sphinx, XML, etc. Would it be convenient to start with some very simple text-like format that easily converts to the formats listed above, and then at some later stage eventually go with a particular format?&lt;/paragraph&gt;&lt;/list_item&gt;&lt;list_item&gt;&lt;paragraph&gt;Do you fi nd it problematic that you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one place, and include it anywhere?&lt;/paragraph&gt;&lt;/list_item&gt;&lt;/bullet_list&gt;&lt;/field_bod y&gt;&lt;/field&gt;&lt;/field_list&gt;&lt;paragraph&gt;If any of these questions are of interest, you should keep on reading.&lt;/paragraph&gt;&lt;/section&gt;&lt;section ids="the-doconce-concept" names="the\ doconce\ concept"&gt;&lt;title&gt;The Doconce Concept&lt;/title&gt;&lt;paragraph&gt;Doco nce is two things:&lt;/paragraph&gt;&lt;block_quote&gt;&lt;enumerated_list enumtype="arabic" pr efix="" suffix="."&gt;&lt;list_item&gt;&lt;paragraph&gt;Doconce is a very simple and minimally tagged markup language that look like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, Wiki, LaTeX, PDF, reStructuredText (reST), Sphinx, Epytext, and also plain text (where non-obvious formatting/tags are removed for clear reading in, e.g., emails). From reStructuredText you can go to XML, HTML, LaTeX, PDF, OpenOffice, and from the latter to RTF and MS Word.&lt;/paragraph&gt;&lt;/list_item&gt;&lt;list_item&gt;&lt;paragraph&gt;Doconce is a working strategy for never duplicating information. Text is written in a single place and then transformed to a number of different destinations of diverse type (software source code, manuals, tutorials, books, wikis, memos, emails, etc.). The Doconce markup language support this working strategy. The slogan is: "Write once, include anywhere"&lt;/paragraph&gt;&lt;/list_item&gt; &lt;/enumerated_list&gt;&lt;/block_quote&gt;&lt;paragraph&gt;A wide range of markup languages exis t. For example, reStructuredText and Sphinx have recently become popular. So why another one?&lt;/paragraph&gt;&lt;block_quote&gt;&lt;bulle t_list bullet="*"&gt;&lt;list_item&gt;&lt;paragraph&gt;Doconce can convert to plain &lt;emphasis&gt;u ntagged&lt;/emphasis&gt; text, more desirable for computer programs and email.&lt;/paragraph&gt;&lt;/list_item&gt;&lt;list_ite m&gt;&lt;paragraph&gt;Doconce has less cluttered tagging of text.&lt;/paragraph&gt;&lt;/list_item&gt; &lt;list_item&gt;&lt;paragraph&gt;Doconce has better support for copying in parts of compute r code, say in examples, directly from the source code files.&lt;/paragraph&gt;&lt;/list_item&gt;&lt;li st_item&gt;&lt;paragraph&gt;Doconce has stronger support for mathematical typesetting, an d has many features for being integrated with (big) LaTeX projects.&lt;/paragraph&gt;&lt;/l ist_item&gt;&lt;list_item&gt;&lt;paragraph&gt;Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google Wiki, LaTeX, and Sphinx. A primary application of Doconce </pre>	



## tutorial.xml

```

is just to make the initial versions of a Sphinx or Wiki document.</paragraph></
list_item></bullet_list></block_quote><paragraph>Doconce was particularly writte
n for the following sample applications:</paragraph><block_quote><bullet_list bu
llet="*"><list_item><paragraph>Large books written in LaTeX, but where many piec
es (computer demos,
projects, examples) can be written in Doconce to appear in other
contexts in other formats, including plain HTML, Sphinx, or MS Word.</paragraph>
</list_item><list_item><paragraph>Software documentation, primarily Python doc s
trings, which one wants
to appear as plain untagged text for viewing in Pydoc, as reStructuredText
for use with Sphinx, as wiki text when publishing the software at
googlecode.com, and as LaTeX integrated in, e.g., a thesis.</paragraph></list_it
em><list_item><paragraph>Quick memos, which start as plain text in email, then s
ome small
amount of Doconce tagging is added, before the memos can appear as
MS Word documents or in wikis.</paragraph></list_item></bullet_list></block_quot
e><paragraph>Disclaimer: Doconce is a simple tool, largely based on interpreting
and handling text through regular expressions. The possibility for
tweaking the layout is obviously limited since the text can go to
all sorts of sophisticated markup languages. Moreover, because of
limitations of regular expressions, some formatting may face problems
when transformed to other formats.</paragraph></section><section ids="what-does-
doconce-look-like" names="what\ does\ doconce\ look\ like?"><title>What Does Doc
once Look Like?</title><paragraph>Doconce text looks like ordinary text, but the
re are some almost invisible
text constructions that allow you to control the formating. For example,</paragr
aph><block_quote><bullet_list bullet="*"><list_item><paragraph>bullet lists aris
e from lines starting with an asterisk,</paragraph></list_item><list_item><parag
raph><emphasis>emphasized words</emphasis> are surrounded by asterisks,</paragra
ph></list_item><list_item><paragraph><strong>words in boldface</strong> are surr
ounded by underscores,</paragraph></list_item><list_item><paragraph>words from c
omputer code are enclosed in back quotes and
then typeset verbatim,</paragraph></list_item><list_item><paragraph>blocks of co
mputer code can easily be included, also from source files,</paragraph></list_it
em><list_item><paragraph>blocks of LaTeX mathematics can easily be included,</pa
ragraph></list_item><list_item><paragraph>there is support for both LaTeX and te
xt-like inline mathematics,</paragraph></list_item><list_item><paragraph>figures
with captions, URLs with links, labels and references
are supported,</paragraph></list_item><list_item><paragraph>comments can be inse
rted throughout the text,</paragraph></list_item><list_item><paragraph>with a si
mple preprocessor, which is integrated, one can include
other documents (files) and large portions of text can be defined
in or out of the text,</paragraph></list_item><list_item><paragraph>with the Mak
o preprocessor one can even embed Python
code and use this to steer generation of Doconce text.</paragraph></list_item></
bullet_list></block_quote><paragraph>Here is an example of some simple text writ
ten in the Doconce format:</paragraph><literal_block xml:space="preserve">====
A Subsection with Sample Text ====
label{my:first:sec}

Ordinary text looks like ordinary text, and the tags used for
_<b>boldface</b>_ 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,

```

## tutorial.xml

- o item 1
- o item 2
- o item 3

URLs with a link word are possible, as in `<reference name="hpl" refuri="http://folk.uio.no/hpl">hpl</reference>`.

If the word is URL, the URL itself becomes the link name, as in `<reference name="URL" refuri="tutorial.do.txt">URL</reference>`.

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

Doconce also allows inline comments such as `[hpl: here I will make some remarks to the text]` for allowing authors to make notes. Inline comments can be removed from the output by a command-line argument (see Chapter `ref{doconce2formats}` for an example).

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines

The Doconce text above results in the following little document:

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

```
<strong>boldface</strong> words, <emphasis>emphasized</emphasis> words, and <literal>computer</literal> words look
```

```
natural in plain text. Lists are typeset as you would do in an email,
```

```
<block_quote>
<bullet_list bullet="*">
<list_item><paragraph>item 1</paragraph></list_item>
<list_item><paragraph>item 2</paragraph></list_item>
<list_item><paragraph>item 3</paragraph></list_item>
</bullet_list>
</block_quote>
<paragraph>Lists can also have numbered items instead of bullets, just use an <literal>o</literal>
```

```
>
(for ordered) instead of the asterisk:
<block_quote>
<enumerated_list enumtype="arabic" prefix="" suffix=".">
<list_item><paragraph>item 1</paragraph></list_item>
<list_item><paragraph>item 2</paragraph></list_item>
<list_item><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" refuri="http://folk.uio.no/hpl"/>.
```

```
If the word is URL, the URL itself becomes the link name,
as in <reference name="tutorial.do.txt" refuri="tutorial.do.txt">tutorial.do.txt</reference>
<target ids="tutorial-do-txt" names="tutorial.do.txt" refuri="tutorial.do.txt"/>.
</paragraph>
<paragraph>References to sections may use logical names as labels (e.g., a
```

```
<reference name="label" refid="my:first:sec">label</reference> command right after the section title), as in the reference to
the chapter <reference name="A Subsection with Sample Text" refid="a-subsection-with-sample-text">A Subsection with Sample Text</reference>.
</paragraph>
<paragraph>Doconce also allows inline comments such as (<strong>hpl</strong>: here I wil
```

## tutorial.xml

```

l make
some remarks to the text) for allowing authors to make notes. Inline
comments can be removed from the output by a command-line argument
(see the chapter <reference name="From Doconce to Other Formats" refid="from-doc
once-to-other-formats">From Doconce to Other Formats</reference> for an example)
.</paragraph><paragraph>Tables are also supported, e.g.,</paragraph><table><tgro
up cols="3"><colspec colwidth="12"/><colspec colwidth="12"/><colspec colwidth="1
2"/><thead><row><entry><paragraph>time</paragraph></entry><entry><paragraph>velo
city</paragraph></entry><entry><paragraph>acceleration</paragraph></entry></row>
</thead><tbody><row><entry><paragraph>0.0</paragraph></entry><entry><paragraph>1
.4186</paragraph></entry><entry><paragraph>-5.01</paragraph></entry></row><row><
entry><paragraph>2.0</paragraph></entry><entry><paragraph>1.376512</paragraph></
entry><entry><paragraph>11.919</paragraph></entry></row><row><entry><paragraph>4
.0</paragraph></entry><entry><paragraph>1.1E+1</paragraph></entry><entry><paragr
aph>14.717624</paragraph></entry></row></tbody></tgroup></table></section><secti
on ids="mathematics-and-computer-code" names="mathematics\ and\ computer\ code">
<title>Mathematics and Computer Code</title><paragraph>Inline mathematics, such
as  $v = \sin(x)$ ,
allows the formula to be specified both as LaTeX and as plain text.
This results in a professional LaTeX typesetting, but in other formats
the text version normally looks better than raw LaTeX mathematics with
backslashes. An inline formula like  $v = \sin(x)$  is
typeset as:</paragraph><literal_block xml:space="preserve">\nu = \sin(x)$| $v =
\sin(x)$</literal_block><paragraph>The pipe symbol acts as a delimiter between La
TeX code and the plain text
version of the formula.</paragraph><paragraph>Blocks of mathematics are better t
ypeset with raw LaTeX, inside
<literal>!bt</literal> and <literal>!et</literal> (begin tex / end tex) instruct
ions.
The result looks like this:</paragraph><literal_block xml:space="preserve">\begi
n{eqnarray}
{\partial u\over\partial t} \&=\& \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>It is po
ssible to add a specification of a (ptex2tex-style)
environment for typesetting the verbatim code block, e.g., <literal>!bc xxx</lit
eral>
where <literal>xxx</literal> is an identifier like <literal>pycod</literal> for
code snippet in Python,
<literal>sys</literal> for terminal session, etc. When Doconce is filtered to La
TeX,
these identifiers are used as in ptex2tex and defined in a
configuration file <literal>.ptext2tex.cfg</literal>, while when filtering
to Sphinx, one can have a comment line in the Doconce file for
mapping the identifiers to legal language names for Sphinx (which equals
the legal language names for Pygments):</paragraph><literal_block xml:space="pre
serve"># sphinx code-blocks: pycod=python cod=py cppcod=c++ sys=console</literal

```

## tutorial.xml

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

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

One can also copy computer code directly from files, either the complete file or specified parts. Computer code is then never duplicated in the documentation (important for the principle of avoiding copying information!). A complete file is typeset with `!bc pro`, while a part of a file is copied into a `!bc cod` environment. What `pro` and `cod` mean is then defined through a `.ptex2tex.cfg` file for LaTeX and a `sphinx code-blocks` comment for Sphinx.

Another document can be included by writing `#include "mynote.do.txt"` on a line starting with (another) hash sign. Doconce documents have extension `.do.txt`. The `do` part stands for doconce, while the trailing `.txt` denotes a text document so that editors gives you the right writing environment for plain text.

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

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

Recent versions of Doconce also offer cross referencing, typically one can define labels below (sub)sections, in figure captions, or in

## tutorial.xml

equations, and then refer to these later. Entries in an index can be defined and result in an index at the end for the LaTeX and Sphinx formats. Citations to literature, with an accompanying bibliography in a file, are also supported. The syntax of labels, references, citations, and the bibliography closely resembles that of LaTeX, making it easy for Doconce documents to be integrated in LaTeX projects (manuals, books). For further details on functionality and syntax we refer to the `<literal>docs/manual/manual.do.txt</literal>` file (see the

```
<reference name="demo page" refuri="https://doconce.googlecode.com/hg/trunk/docs/demos/manual/index.html">demo
page</reference><target ids="demo-page" names="demo\ page" refuri="https://doconce.googlecode.com/hg/trunk/docs/demos/manual/index.html"/> for various formats of this document).</paragraph><comment xml:space="preserve">Example on including another Doconce file (using preprocess):</comment><target refid="doconce2formats"/></section></section><section ids="from-doconce-to-other-formats doconce2formats" names="from\ doconce\ to\ other\ formats doconce2formats"><title>From Doconce to Other Formats</title><paragraph>Transformation of a Doconce document to various other formats applies the script <literal>doconce format</literal>:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce format mydoc.do.txt</literal_block><paragraph>The <literal>preprocess</literal> program is always used to preprocess the file first, and options to <literal>preprocess</literal> can be added after the filename. For example:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce format LaTeX mydoc.do.txt -Dextra_sections</literal_block><paragraph>The variable <literal>FORMAT</literal> is always defined as the current format when running <literal>preprocess</literal>. That is, in the last example, <literal>FORMAT</literal> is defined as <literal>LaTeX</literal>. Inside the Doconce document one can then perform format specific actions through tests like <literal>#if FORMAT == &quot;LaTeX&quot;ot;</literal>.</paragraph><paragraph>Inline comments in the text are removed from the output by:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce format LaTeX mydoc.do.txt remove_inline_comments</literal_block><paragraph>One can also remove such comments from the original Doconce file by running a helper script in the <literal>bin</literal> folder of the Doconce source code:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce remove_inline_comments mydoc.do.txt</literal_block><paragraph>This action is convenient when a Doconce document reaches its final form.</paragraph><section ids="html" names="html"><title>HTML</title><paragraph>Making an HTML version of a Doconce file <literal>mydoc.do.txt</literal> is performed by:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce format HTML mydoc.do.txt</literal_block><paragraph>The resulting file <literal>mydoc.html</literal> can be loaded into any web browser for viewing.</paragraph></section><section ids="latex" names="latex"><title>LaTeX</title><paragraph>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.</paragraph><definition_list><definition_list_item><term><emphasis>Step 1.</emphasis> Filter the doconce text to a pre-LaTeX form <literal>mydoc.p.tex</literal> for</term><definition><paragraph><literal>ptex2tex</literal>:</paragraph><literal_block xml:space="preserve">Unix/DOS> doconce format LaTeX mydoc.do.txt</literal_block></definition></definition_list_item></definition_list><paragraph>LaTeX-specific commands (&quot;newcommands&quot;) in math formulas and similar can be placed in files <literal>newcommands.tex</literal>, <literal>newcommands_keep.tex</literal>, or

```

## tutorial.xml

`<literal>newcommands_replace.tex</literal>` (see the section `<reference name="Macros (Newcommands), Cross-References, Index, and Bibliography" refid="macros-newcommands-cross-references-index-and-bibliography">Macros (Newcommands), Cross-References, Index, and Bibliography</reference>`).

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

Run `<literal>ptex2tex</literal>` (if you have it) to make a standard LaTeX file:

```
Unix/DOS&gt; ptex2tex mydoc
```

or just perform a plain copy:

```
Unix/DOS&gt; cp mydoc.p.tex mydoc.tex
```

Doconce generates a `<literal>.p.tex</literal>` file with some preprocessor macros.

For example, to enable font Helvetica instead of the standard Computer Modern font:

```
Unix/DOS&gt; ptex2tex -DHELVETICA mydoc
```

The title, authors, and date are by default typeset in a non-standard

way to enable a nicer treatment of multiple authors having institutions in common. The standard LaTeX `"maketitle"` heading is also available through:

```
Unix/DOS&gt; ptex2tex -DTRAD_LATEX_HEADING mydoc
```

The `<literal>ptex2tex</literal>` tool makes it possible to easily switch between many different fancy formattings of computer or verbatim code in LaTeX documents. After any `<literal>!bc sys</literal>` command in the Doconce source you can

insert verbatim block styles as defined in your `<literal>.ptex2tex.cfg</literal>` file, e.g., `<literal>!bc sys cod</literal>` for a code snippet, where `<literal>cod</literal>` is set to a certain environment in `<literal>.ptex2tex.cfg</literal>` (e.g., `<literal>CodeIntended</literal>`).

There are over 30 styles to choose from.

Compile `<literal>mydoc.tex</literal>`

and create the PDF file:

```
Unix/DOS&gt; latex mydoc
```

```
Unix/DOS&gt; latex mydoc
Unix/DOS&gt; makeindex mydoc    # if index
Unix/DOS&gt; bibitem mydoc      # if bibliography
Unix/DOS&gt; latex mydoc
```

Unix/DOS&gt; dvipdf mydoc

If one wishes to use the `<literal>Minted_Python</literal>`, `<literal>Minted_Cpp</literal>`, etc., environments in `<literal>ptex2tex</literal>` for typesetting code, the `<literal>minted</literal>` LaTeX package is needed.

This package is included by running `<literal>doconce format</literal>` with the `<literal>-DMINTED</literal>` option:

```
Unix/DOS&gt; ptex2tex -DMINTED mydoc
```

In this case,

`<literal>latex</literal>` must be run with the

```
Unix/DOS&gt; latex -shell-escape mydoc
```

```
Unix/DOS&gt; latex -shell-escape mydoc
Unix/DOS&gt; makeindex mydoc    # if index
Unix/DOS&gt; bibitem mydoc      # if bibliography
Unix/DOS&gt; latex -shell-escape mydoc
```

Unix/DOS&gt; dvipdf mydoc

The `<literal>-shell-escape</literal>` option is required because the `<literal>minted.sty</literal>` style file runs the `<literal>pygments</literal>` program to format code, and this program

cannot be run from `<literal>latex</literal>` without the `<literal>-shell-escape</literal>` option.

`<section ids="plain-ascii-text" names="plain\ ascii\ text"><title>Plain ASCII Text</title><paragraph>We can go from Doconce "back to" plain untagged text suitable for viewing`

## tutorial.xml

```

in terminal windows, inclusion in email text, or for insertion in
computer source code:</paragraph><literal_block xml:space="preserve">Unix/DOS&gt;
; doconce format plain mydoc.do.txt # results in mydoc.txt</literal_block></sec
tion><section ids="restructuredtext" names="restructuredtext"><title>reStructure
dText</title><paragraph>Going from Doconce to reStructuredText gives a lot of po
ssibilities to
go to other formats. First we filter the Doconce text to a
reStructuredText file <literal>mydoc.rst</literal>:</paragraph><literal_block xm
l:space="preserve">Unix/DOS&gt; doconce format rst mydoc.do.txt</literal_block><
paragraph>We may now produce various other formats:</paragraph><literal_block xm
l:space="preserve">Unix/DOS&gt; rst2html.py mydoc.rst &gt; mydoc.html # HTML
Unix/DOS&gt; rst2latex.py mydoc.rst &gt; mydoc.tex # LaTeX
Unix/DOS&gt; rst2xml.py mydoc.rst &gt; mydoc.xml # XML
Unix/DOS&gt; rst2odt.py mydoc.rst &gt; mydoc.odt # OpenOffice</literal_block>
<paragraph>The OpenOffice file <literal>mydoc.odt</literal> can be loaded into O
penOffice and
saved in, among other things, the RTF format or the Microsoft Word format.
That is, one can easily go from Doconce to Microsoft Word.</paragraph></section>
<section dupnames="sphinx" ids="sphinx"><title>Sphinx</title><paragraph>Sphinx d
ocuments can be created from a Doconce source in a few steps.</paragraph><paragr
aph><emphasis>Step 1.</emphasis> Translate Doconce into the Sphinx dialect of
the reStructuredText format:</paragraph><literal_block xml:space="preserve">Unix
/DOS&gt; doconce format sphinx mydoc.do.txt</literal_block><paragraph><emphasis>
Step 2.</emphasis> Create a Sphinx root directory with a <literal>conf.py</liter
al> file,
either manually or by using the interactive <literal>sphinx-quickstart</literal>
program. Here is a scripted version of the steps with the latter:</paragraph><li
teral_block xml:space="preserve">mkdir sphinx-rootdir
sphinx-quickstart &lt;&lt;EOF
sphinx-rootdir
n
-
Name of My Sphinx Document
Author
version
version
.rst
index
n
y
n
n
n
n
n
y
n
n
y
y
y
EOF</literal_block><paragraph>These statements are automated by the command:</pa
ragraph><literal_block xml:space="preserve">Unix/DOS&gt; doconce sphinx_dir mydo
c.do.txt</literal_block><paragraph><emphasis>Step 3.</emphasis> Move the <litera
l>tutorial.rst</literal> file to the Sphinx root directory:</paragraph><literal_
block xml:space="preserve">Unix/DOS&gt; mv mydoc.rst sphinx-rootdir</literal_blo
ck><paragraph>If you have figures in your document, the relative paths to those
will
be invalid when you work with <literal>mydoc.rst</literal> in the <literal>sphin
x-rootdir</literal>

```

## tutorial.xml

directory. Either edit `<literal>mydoc.rst</literal>` so that figure file paths are correct, or simply copy your figure directory to `<literal>sphinx-rootdir</literal>` (if all figures are located in a subdirectory).

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

```
toctree::
    :maxdepth: 2

    mydoc
```

(The spaces before `<literal>mydoc</literal>` 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
```

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

Desired extension: sphinx can utilize a `"pycod"` or `"c++cod"`; instructions as currently done in latex for ptex2tex and write out the right code block name accordingly.

**Google Code Wiki**

There are several different wiki dialects, but Doconce only supports the one used by [Google Code](http://code.google.com/p/support/wiki/WikiSyntax) (`<reference name="Google Code" refuri="http://code.google.com/p/support/wiki/WikiSyntax">Google Code</reference>` `<target ids="google-code" names="google\ code" refuri="http://code.google.com/p/support/wiki/WikiSyntax"/>`). The transformation to this format, called `<literal>gwiki</literal>` to explicitly mark it as the Google Code dialect, is done by:

```
Unix/DOS> doconce format gwiki mydoc.do.txt
```

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

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

**Tweaking the Doconce Output**

Occasionally, one would like to tweak the output in a certain format from Doconce. One example is figure filenames when transforming Doconce to reStructuredText. Since Doconce does not know if the `<literal>.rst</literal>` file is going to be filtered to LaTeX or HTML, it cannot know if `<literal>.eps</literal>` or `<literal>.png</literal>` is the most appropriate image filename.



## tutorial.xml

The solution is to use a text substitution command or code with, e.g., sed, perl, python, or scitools subst, to automatically edit the output file from Doconce. It is then wise to run Doconce and the editing commands from a script to automate all steps in going from Doconce to the final format(s). The `<literal>make.sh</literal>` files in `<literal>docs/manual</literal>` and `<literal>docs/tutorial</literal>` constitute comprehensive examples on how such scripts can be made.

The current text is generated from a Doconce format stored in the file: `<literal_block xml:space="preserve">docs/tutorial/tutorial.do.txt</literal_block>`

The file `<literal>make.sh</literal>` in the `<literal>tutorial</literal>` directory of the Doconce source code contains a demo of how to produce a variety of formats. The source of this tutorial, `<literal>tutorial.do.txt</literal>` is the starting point. Running `<literal>make.sh</literal>` and studying the various generated files and comparing them with the original `<literal>tutorial.do.txt</literal>` file, gives a quick introduction to how Doconce is used in a real case.

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

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

**Dependencies**

If you make use of preprocessor directives in the Doconce source, either [Preprocess](http://code.google.com/p/preprocess) or [Mako](http://www.makotemplates.org) must be installed. To make LaTeX documents (without going through the reStructuredText format) you also need [ptex2tex](http://code.google.com/p/ptex2tex) and some style files that `<literal>ptex2tex</literal>` potentially makes use of. Going from reStructuredText to formats such as XML, OpenOffice, HTML, and LaTeX requires [docutils](http://docutils.sourceforge.net). Making Sphinx documents requires of course [Sphinx](http://sphinx.pocoo.org).

All of the mentioned potential dependencies are pure Python packages which are easily installed.

**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,

”

## tutorial.xml

”

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>