Doconce Description

Hans Petter Langtangen[2,1]

[1]University of Oslo

[2]Simula Research Laboratory

Mar 26, 2011

1 What Is Doconce?

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.

You can jump to Section 2.2 to see a recipe for how to use Doconce, unless you need some more motivation for the problem which Doconce tries to solve.

2 Motivation: Problems with Documenting Software

Duplicated Information. It is common to write some software documentation in the code (doc strings in Python, doxygen in C++, javadoc in Java) while similar documentation is often also included in a LaTeX or HTML manual or tutorial. Although the various types of documentation may start out to be the same, different physical files must be used since very different tagging is required for different output formats. Over time the duplicated information starts to diverge. Severe problems with such unsynchronized documentation was one motivation for developing the Doconce concept and tool.

Tagging Issues in Python Documentation. A problem with doc strings in Python is that they benefit greatly from some tagging, Epytext or reStructuredText, when transformed to HTML or PDF manuals. However, such tagging looks annoying in Pydoc, which just shows the pure doc string. For Pydoc we should have more minimal (or no) tagging (students and newbies are in particular annoyed by any unfamiliar tagging of ASCII text). On the contrary, manuals or tutorials in HTML and LaTeX need quite much tagging.

Solution. Accurate information is crucial and can only be maintained in a *single physical* place (file), which must be converted (filtered) to suitable formats and included in various documents (HTML/LaTeX manuals/tutorials, Pydoc/Epydoc/HappyDoc reference manuals).

A Common Format. There is no existing format and associated conversion tools that allow a "singleton" documentation file to be filtered to LaTeX, HTML, XML, PDF, Epydoc, HappyDoc, Pydoc, *and* plain untagged text. As we are involved with mathematical software, the LaTeX manuals should have nicely typeset mathematics, while Pydoc, Epydoc, and HappyDoc must show LaTeX math in verbatim mode. Unfortunately, Epytext is annoyed by even very simple LaTeX math (also in verbatim environments). To summarize, we need

1. A minimally tagged markup language with full support for for mathematics and verbatim computer code.

2. Filters for producing highly tagged formats (LaTeX, HTML, XML), medium tagged formats (reStructuredText, Epytext), and plain text with completely invivisble tagging.

3. Tools for inserting appropriately filtered versions of a "singleton" documentation file in other documents (manuals, tutorials, doc strings).

One answer to these points is the Doconce markup language, its associated tools, and a C-style preprocessor tool or the Mako template system. Then we can *write once, include anywhere*! And what we write is close to plain ASCII text.

But isn't reStructuredText exactly the format that fulfills the needs above? Yes and no. Yes, because reStructuredText can be filtered to a lot of the mentioned formats. No, because of the reasons listed in Section 1, but perhaps the strongest feature of Doconce is that it integrates well with LaTeX: Large LaTeX documents (book) can be made of many smaller Doconce units, typically describing examples and computer codes, glued with mathematical pieces written entirely in LaTeX and with heavy cross-referencing of equations, as is usual in mathematical texts. All the Doconce units can then be available also as standalone examples in wikis or Sphinx pages and thereby used in other occasions (including software documentation and teaching material). This is a promising way of composing future books of units that can be reused in many contexts and formats, currently being explored by the Doconce maintainer.

A final warning may be necessary: The Doconce format is a minimalistic formatting language. It is ideal when you start a new project when you are uncertain about which format to choose. At some later stage, when you need quite some sophisticated formatting and layout, you can perform the final filtering of Doconce into something more appropriate for future demands. The convenient thing is that the format decision can be posponed (maybe forever - which is the common experience of the Doconce developer).

2.1 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.

2.2 The Doconce Software Documentation Strategy

- Write software documentation, both tutorials and manuals, in the Doconce format. Use many files - and never duplicate information!
- Use `#include` statements in source code (especially in doc strings) and in LaTeX documents for including documentation files. These documentation files must be filtered to an appropriate format by the program `doconce` before being included. In a Python context, this means plain text for computer source code (and Pydoc); Epytext for Epydoc API documentation, or the Sphinx dialect of reStructuredText for Sphinx API documentation; LaTeX for LaTeX manuals; and possibly reStructuredText for XML, Docbook, OpenOffice, RTF, Word.
- Run the preprocessor `preprocess` on the files to produce native files for pure computer code and for various other documents.

Consider an example involving a Python module in a `basename.p.py` file. The `.p.py` extension identifies this as a file that has to be preprocessed) by the `preprocess` program. In a doc string in `basename.p.py` we do a preprocessor include in a comment line, say (use triple quotes in the doc string in case the `doc1` documentation includes code snippets with doc strings with the usual triple double quotes): `#  #include docstrings/doc1.dst.txt`. The file `docstrings/doc1.dst.txt` is a file filtered to a specific format (typically plain text, reStructedText, or Epytext) from an original "singleton" documentation file named `docstrings/doc1.do.txt`. The `.dst.txt` is the extension of a file filtered ready for being included in a doc string (`d` for doc, `st` for string).

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

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

steps are automated by the `insertdocstr.py` script. Here are the corresponding Unix commands:

Terminal

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

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

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

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

2.3 Demos

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

Terminal

```
docs/manual/manual.do.txt
```

file in the Doconce source code tree. We have made a demo web page where you can compare the Doconce source with the output in many different formats: HTML, LaTeX, plain text, etc.

The file `make.sh` in the same directory as the `manual.do.txt` file (the current text) shows how to run `doconce format` on the Doconce file to obtain documents in various formats.

Another demo is found in

Terminal

docs/tutorial/tutorial.do.txt

In the tutorial directory there is also a make.sh file producing a lot of formats, with a corresponding web demo of the results.

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 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`:

Terminal

```
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 Section 5.8). If these files are present, they are included in the LaTeX document so that your commands are defined.

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

Terminal

```
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> 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:

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> 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.

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

3.6 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

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 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 The Doconce Markup Language

The Doconce format introduces four constructs to markup text: lists, special lines, inline tags, and environments.

4.1 Lists

An unordered bullet list makes use of the `*` as bullet sign and is indented as follows

```
* item 1

* item 2

  * subitem 1, if there are more
    lines, each line must
    be intended as shown here

  * subitem 2,
    also spans two lines

* item 3
```

This list gets typeset as

- item 1
- item 2
  - subitem 1, if there are more lines, each line must be intended as shown here
  - subitem 2, also spans two lines
- item 3

In an ordered list, each item starts with an `o` (as the first letter in "ordered"):

```
o item 1
o item 2
  * subitem 1
  * subitem 2
o item 3
```

resulting in

1. item 1
2. item 2
   - subitem 1
   - subitem 2
3. item 3

Ordered lists cannot have an ordered sublist, i.e., the ordering applies to the outer list only.

In a description list, each item is recognized by a dash followed by a keyword followed by a colon:

```
- keyword1: explanation of keyword1
- keyword2: explanation
  of keyword2 (remember to indent properly
  if there are multiple lines)
```

The result becomes

keyword1: explanation of keyword1

keyword2: explanation of keyword2 (remember to indent properly if there are multiple lines)

4.2 Special Lines

The Doconce markup language has a concept called *special lines*. Such lines starts with a markup at the very beginning of the line and are used to mark document title, authors, date, sections, subsections, paragraphs., figures, etc.

Heading with Title and Author(s). Lines starting with `TITLE:`, `AUTHOR:`, and `DATE:` are optional and used to identify a title of the document, the authors, and the date. The title is treated as the rest of the line, so is the date, but the author text consists of the name and associated institution(s) with the syntax

```
name at institution1 and institution2 and institution3
```

The `at` with surrounding spaces is essential for adding information about institution(s) to the author name, and the `and` with surrounding spaces is essential as delimiter between different institutions. Multiple authors require multiple `AUTHOR:` lines. All information associated with `TITLE:` and `AUTHOR:` keywords must appear on a single line. Here is an example:

```
TITLE: On an Ultimate Markup Language
AUTHOR: H. P. Langtangen at Center for Biomedical Computing, Simula Research Laboratory and Dept.
AUTHOR: Kaare Dump at Segfault, Cyberspace Inc.
AUTHOR: A. Dummy Author
DATE: November 9, 2016
```

Note the how one can specify a single institution, multiple institutions, and no institution. In some formats (including reStructuredText and Sphinx) only the author names appear. Some formats have "intelligence" in listing authors and institutions, e.g., the plain text format:

```
Hans Petter Langtangen [1, 2]
Kaare Dump [3]
A. Dummy Author

[1] Center for Biomedical Computing, Simula Research Laboratory
[2] Department of Informatics, University of Oslo
[3] Segfault, Cyberspace Inc.
```

Similar typesetting is done for LaTeX and HTML formats.

Section Headings. Section headings are recognized by being surrounded by equal signs (=) or underscores before and after the text of the headline. Different section levels are recognized by the associated number of underscores or equal signs (=):

- 7 underscores or equal signs for sections
- 5 for subsections
- 3 for subsubsections
- 2 underscrores (only! - it looks best) for paragraphs (paragraph heading will be inlined)

Headings can be surrounded by blanks if desired.

Here are some examples:

```
======= Example on a Section Heading =======

The running text goes here.

      ===== Example on a Subsection Heading =====
The running text goes here.

          ===Example on a Subsubsection Heading===

The running text goes here.

__A Paragraph.__ The running text goes here.
```

The result for the present format looks like this:

5 Example on a Section Heading

The running text goes here.

5.1 Example on a Subsection Heading

The running text goes here.

Example on a Subsubsection Heading. The running text goes here.

A Paragraph. The running text goes here.

Figures. Figures are recognized by the special line syntax

```
FIGURE:[filename, height=xxx width=yyy scale=zzz] possible caption
```

The filename can be without extension, and Doconce will search for an appropriate file with the right extension. If the extension is wrong, say `.eps` when requesting an HTML format, Doconce tries to find another file, and if not, the given file is converted to a proper format (using ImageMagick's `convert` utility).

The height, width, and scale keywords (and others) can be included if desired and may have effect for some formats. Note the comma between the sespecifications and that there should be no space around the = sign.

Note also that, like for `TITLE:` and `AUTHOR:` lines, all information related to a figure line must be written on the same line. Introducing newlines in a long caption will destroy the formatting (only the part of the caption appearing on the same line as `FIGURE:` will be included in the formatted caption).

Movies. Here is an example on the `MOVIE:` keyword for embedding movies. This feature works only for the `LaTeX` and `HTML` formats.

```
MOVIE: [filename, height=xxx width=yyy] possible caption
```

Figure 1: It can't get worse than this....

The LaTeX format results in a file that requires the movie15 package in order to play movies in PDF via Acroread. The HTML format will play the movie right away, while for all other formats there is no movie support. The HTML format can also treat filenames of the form `myframes*.png`. In that case, a player for showing the sequence of frames is inserted in the HTML file.

Computer Code. Another type of special lines starts with `@@@CODE` and enables copying of computer code from a file directly into a verbatim environment, see Section 5.6 below.

5.2 Inline Tagging

Doconce supports tags for *emphasized phrases*, **boldface phrases**, and `verbatim text` (also called type writer text, for inline code) plus LaTeX/TeX inline mathematics, such as $\nu = \sin(x)$.

Emphasized text is typeset inside a pair of asterisk, and there should be no spaces between an asterisk and the emphasized text, as in

```
*emphasized words*
```

Boldface font is recognized by an underscore instead of an asterisk:

```
_several words in boldface_ followed by *ephasized text*.
```