

# Doconce: Document Once, Include Anywhere

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

<sup>1</sup>Simula Research Laboratory

<sup>2</sup>University of Oslo

Dec 11, 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 need to write documents in varying formats but find it difficult to remember all the typesetting details of various formats like [LaTeX](#), [HTML](#), [reStructuredText](#), [Sphinx](#), and [wiki](#)? Would it be convenient to generate the typesetting details of a particular format from a very simple text-like format with minimal tagging?
- Do you have the same information scattered around in different documents in different typesetting formats? Would it be a good idea to write things once, in one format, stored in one place, and include it anywhere?

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

## 1 The Doconce Concept

Doconce is two things:

1. Doconce is a very simple and minimally tagged markup language that looks like ordinary ASCII text (much like what you would use in an email), but the text can be transformed to numerous other formats, including HTML, 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. (An experimental translator to Pandoc is under development, and from Pandoc one can generate Markdown, reST,  $\LaTeX$ , HTML, PDF, DocBook XML, OpenOffice, GNU Texinfo, MediaWiki, RTF, Groff, and other formats.)

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

Here are some Doconce features:

- Doconce markup does include tags, so the format is more tagged than Markdown and Pandoc, but less than reST, and very much less than  $\LaTeX$  and HTML.
- Doconce can be converted to plain *untagged* text, often desirable for computer programs and email.
- Doconce has good support for copying in parts of computer code, say in examples, directly from the source code files.
- Doconce has full support for  $\LaTeX$  math, and integrates very well with big  $\LaTeX$  projects (books).
- Doconce is almost self-explanatory and is a handy starting point for generating documents in more complicated markup languages, such as Google wiki,  $\LaTeX$ , and Sphinx. A primary application of Doconce is just to make the initial versions of a Sphinx or wiki document.
- Contrary to the similar Pandoc translator, Doconce integrates with Sphinx and Google wiki. However, if these formats are not of interest, Pandoc is obviously a superior tool.

Doconce was particularly written for the following sample applications:

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

History: Doconce was developed in 2006 at a time when most popular markup languages used quite some tagging. Later, almost untagged markup languages like Markdown and Pandoc became popular. Doconce is not a replacement of Pandoc, which is a considerably more sophisticated project. Moreover,

Doconce was developed mainly to fulfill the needs for a flexible source code base for books with much mathematics and computer code.

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

## 2 What Does Doconce Look Like?

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

- Bullet lists arise from lines starting with an asterisk.
- *Emphasized words* are surrounded by asterisks.
- **Words in boldface** are surrounded by underscores.
- Words from computer code are enclosed in back quotes and then typeset `verbatim` (in a monospace font).
- Section headings are recognized by equality (=) signs before and after the title, and the number of = signs indicates the level of the section: 7 for main section, 5 for subsection, and 3 for subsubsection.
- Paragraph headings are recognized by a double underscore before and after the heading.
- The abstract of a document starts with *Abstract* as paragraph heading, and all text up to the next heading makes up the abstract,
- Blocks of computer code can easily be included by placing `bc!` (begin code) and `ec!` (end code) commands at separate lines before and after the code block.
- Blocks of computer code can also be imported from source files.
- Blocks of  $\text{\LaTeX}$  mathematics can easily be included by placing `bt!` (begin TeX) and `et!` (end TeX) commands at separate lines before and after the math block.
- There is support for both  $\text{\LaTeX}$  and text-like inline mathematics.
- Figures and movies with captions, simple tables, URLs with links, index list, labels and references are supported.
- Invisible comments in the output format can be inserted throughout the text (`#` at the beginning of a line).

- Visible comments can be inserted so that authors and readers can comment upon the text (and at any time turn on/off output of such comments).
- There is special support for advanced exercises features.
- With a simple preprocessor, Preprocess or Mako, one can include other documents (files) and large portions of text can be defined in or out of the text.
- With the Mako preprocessor one can even embed Python code and use this to steer generation of Doconce text.

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

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

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

    * item 1
    * item 2
    * item 3

Lists can also have automatically numbered items instead of bullets,

    o item 1
    o item 2
    o item 3

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

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

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

Tables are also supported, e.g.,

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

# lines beginning with # are comment lines
```

The Doconce text above results in the following little document:

## 2.1 A Subsection with Sample Text

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

- item 1
- item 2
- item 3

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

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

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

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

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

Tables are also 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

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

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

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

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

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

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

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

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

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

A code block must come after some plain sentence (at least for successful output to `sphinx`, `rst`, and ASCII-close formats), not directly after a section-/paragraph heading or a table.

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

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

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

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

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

Doconce supports a type of macros via a  $\text{\LaTeX}$ -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  $\text{\LaTeX}$  (since  $\text{\LaTeX}$  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  $\text{\LaTeX}$

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  $\text{\LaTeX}$  math text easier to read in the formats that cannot render  $\text{\LaTeX}$ . 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  $\text{\LaTeX}$  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  $\text{\LaTeX}$ , making it easy for Doconce documents to be integrated in  $\text{\LaTeX}$  projects (manuals, books). For further details on functionality and syntax we refer to the `doc/manual/manual.do.txt` file (see the [demo page](#) for various formats of this document).

### 3 From Doconce to Other Formats

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

---

Terminal

---

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

---

or just

---

Terminal

---

```
Terminal> doconce format format mydoc
```

---

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

---

Terminal

---

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

---

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

Inline comments in the text are removed from the output by

---

Terminal

---

```
Terminal> doconce format latex mydoc --skip_inline_comments
```

---

One can also remove all such comments from the original Doconce file by running:

```
Terminal> doconce remove_inline_comments mydoc
```

This action is convenient when a Doconce document reaches its final form and comments by different authors should be removed.

### 3.1 HTML

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

---

Terminal

---

```
Terminal> doconce format html mydoc
```

---

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

### 3.2 L<sup>A</sup>T<sub>E</sub>X

Making a L<sup>A</sup>T<sub>E</sub>X 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

---

```
Terminal> doconce format latex mydoc
```

---

LaTeX-specific commands ("newcommands") in math formulas and similar can be placed in files `newcommands.tex`, `newcommands_keep.tex`, or `newcommands_replace.tex` (see Section 2.3). If these files are present, they are included in the L<sup>A</sup>T<sub>E</sub>X document so that your commands are defined.

**Step 2.** Run `ptex2tex` (if you have it) to make a standard L<sup>A</sup>T<sub>E</sub>X file,

---

Terminal

---

```
Terminal> ptex2tex mydoc
```

---

or just perform a plain copy,



---

Terminal

---

```
Terminal> cp mydoc.p.tex mydoc.tex
```

---

Doconce generates a `.p.tex` file with some preprocessor macros that can be used to steer certain properties of the  $\text{\LaTeX}$  document. For example, to turn on the Helvetica font instead of the standard Computer Modern font, run

---

Terminal

---

```
Terminal> 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. However, the standard  $\text{\LaTeX}$  "maketitle" heading is also available through

---

Terminal

---

```
Terminal> ptex2tex -DLATEX_HEADING=traditional mydoc
```

---

A separate titlepage can be generate by

---

Terminal

---

```
Terminal> ptex2tex -DLATEX_HEADING=titlepage mydoc
```

---

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

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

---

Terminal

---

```
Terminal> latex mydoc
Terminal> latex mydoc
Terminal> makeindex mydoc    # if index
Terminal> bibitem mydoc     # if bibliography
Terminal> latex mydoc
Terminal> dvipdf mydoc
```

---

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

---

Terminal

---

```
Terminal> ptex2tex -DMINTED mydoc
```

---

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

---

Terminal

---

```
Terminal> latex -shell-escape mydoc
Terminal> latex -shell-escape mydoc
Terminal> makeindex mydoc # if index
Terminal> bibitem mydoc # if bibliography
Terminal> latex -shell-escape mydoc
Terminal> dvipdf mydoc
```

---

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

---

```
Terminal> 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

---

```
Terminal> doconce format rst mydoc.do.txt
```

---

We may now produce various other formats:

---

Terminal

---

```
Terminal> rst2html.py mydoc.rst > mydoc.html # html
Terminal> rst2latex.py mydoc.rst > mydoc.tex # latex
Terminal> rst2xml.py mydoc.rst > mydoc.xml # XML
Terminal> rst2odt.py mydoc.rst > mydoc.odt # OpenOffice
```

---

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

### 3.5 Sphinx

Sphinx documents demand quite some steps in their creation. We have automated most of the steps through the `doconce sphinx_dir` command:

---

Terminal

---

```
Terminal> doconce sphinx_dir author="authors' names" \  
          title="some title" version=1.0 dirname=sphinx_dir \  
          theme=mytheme file1 file2 file3 ...
```

---

The keywords `author`, `title`, and `version` are used in the headings of the Sphinx document. By default, `version` is 1.0 and the script will try to deduce authors and title from the doconce files `file1`, `file2`, etc. that together represent the whole document. Note that none of the individual Doconce files `file1`, `file2`, etc. should include the rest as their union makes up the whole document. The default value of `dirname` is `sphinx-rootdir`. The `theme` keyword is used to set the theme for design of HTML output from Sphinx (the default theme is 'default').

With a single-file document in `mydoc.do.txt` one often just runs

---

Terminal

---

```
Terminal> doconce sphinx_dir mydoc
```

---

and then an appropriate Sphinx directory `sphinx-rootdir` is made with relevant files.

The `doconce sphinx_dir` command generates a script `automake-sphinx.sh` for compiling the Sphinx document into an HTML document. This script copies directories named `figs` or `figures` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. If figures or movies are located in other directories, `automake-sphinx.sh` must be edited accordingly. One can either run `automake-sphinx.sh` or perform the steps in the script manually.

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

---

Terminal

---

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

---

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

If it is not desirable to use the autogenerated scripts explained above, here are the complete manual procedure of generating a Sphinx document from a file `mydoc.do.txt`.

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

---

Terminal

---

```
Terminal> doconce format sphinx mydoc
```

---

**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
y
EOF
```

---

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

---

Terminal

---

```
Terminal> cp mydoc.rst sphinx-rootdir
```

---

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

**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
```

---

**Step 6.** View the result:

---

Terminal

---

```
Terminal> firefox _build/html/index.html
```

---

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

---

```
Terminal> 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. If translation to [Pandoc](#) is desired, the Pandoc Haskell program must of course be installed.