DocOnce: Document Once, Include Anywhere

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Jul 20, 2014

- 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 Some DocOnce Features

- Strong support for texts with much math and code.
- Same source can produce a variety of output formats. The following support LaTeX math and (pygmentized) code:
  1. traditional LaTeX B/W documents for printing
  2. color LaTeX PDF documents
  3. color LaTeX PDF documents for viewing on small phones

4. Sphinx HTML documents with 20+ different designs
5. Plain HTML or with a template or another template or solarized
6. HTML for Google or Wordpress blog posts
7. MediaWiki (Wikipedia, Wikibooks, etc)
8. Markdown
9. IPython notebook

Other formats include plain untagged text (for email), Creole wiki (for Bitbucket wikis), Google wiki (for Googlecode), reStructuredText, and Epytext.

- Integration with Mako enables use of variables, functions, if-tests, and loops to parameterize the text and generate various versions of the text for different purposes.
- Computer code can be copied directly from parts of source code files.
- Running text can quickly be edited to slide formats (reveal.js and deck.js, based on HTML5+CSS3).
- Special exercise environments with support for hints, answers, subexercises, etc.
- Automatic inline embedding of YouTube and Vimeo movies.
- Good support for admonitions in various LaTeX and HTML styles for warnings, questions, hints, remarks, summaries, etc.

2 What Does DocOnce Look Like?

DocOnce text looks like ordinary text (much like Markdown), but there are some almost invisible text constructions that allow you to control the formating. Here are some examples.

- Bullet lists automatically arise from lines starting with *, or o if the list is to be enumerated.
- *Emphasized words* are surrounded by *. **Words in boldface** are surrounded by underscores.
- Words from computer code are enclosed in backticks and then typeset `verbatim (in a monospace font)`.
- Section and paragraph headings are recognied special decorating characters (= or _) before and after the heading. The length of the decoration determines the level of the section.

- Blocks of computer code are included by surrounding the blocks with `!bc` (begin code) and `!ec` (end code) tags on separate lines.
- Blocks of computer code can also be imported from source files.
- Blocks of LaTeX mathematics are included by surrounding the blocks with `!bt` (begin TeX) and `!et` (end TeX) tags on separate lines.
- There is support for both LaTeX and text-like inline mathematics such that formulas make sense also when not rendered by LaTeX or MathJax.
- Figures and movies with captions, simple tables, URLs with links, index list, labels and references are supported. YouTube and Vimeo videos are automatically embedded in web documents.
- The abstract of a document starts with *Abstract* as paragraph heading, and all text up to the next heading makes up the abstract,
- Special comment lines are not visible in the output.
- Comments to authors can be inserted throughout the text and made visible or invisible as desired.
- There is an exercise environment with many advanced features.
- With a preprocessor, Preprocess or Mako, one can include other documents (files), large portions of text can be defined in or out of the text, and tailored format-specific constructs can easily be included. With Mako a single text can output its program examples in two or more languages.

2.1 What Can DocOnce Be Used For?

LaTeX is ideal for articles, thesis, and books, but not so suited for web documents. Nice environments for web documents, such as Sphinx, Markdown, or plain HTML, are not particularly well suited for thesis and books. IPython notebooks are ideal for documenting computational experiments, but do not (yet) meet the requirements of books and thesis.

What about migrating a part of a book to blog posts? What about making an MS Word version of parts or an untagged text for inclusion in email? What about efficiently generating slides in modern HTML5 style? DocOnce enables all this with just *one source* (the slogan is *document once - include anywhere*). DocOnce also has many extra features for supporting documents with much code and mathematics, not found in any of the mentioned formats.

2.2 Basic Syntax

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

```
======= First a Section Heading =======

Section headings have 7 equality characters before and after the heading.
With 9 characters we have a chapter heading, 5 gives a subsection
heading, and 3 a subsubsection heading.

===== 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.  Quotations appear inside double backticks
and double single quotes, as in ``this example''.

Lists are typeset as you would do in 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

__Hyperlinks.__ Paragraph headings are surrounded by double underscores.
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". DocOnce distinguishes between paper
and screen output. In traditional paper output, in PDF generated from LaTeX
generated from DocOnce, the URLs of links appear as footnotes.
With screen output, all links are clickable hyperlinks, except in
the plain text format which does not support hyperlinks.

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}. References to equations such as
(ref{myeq1}) work in the same way.

__Inline comments.__
DocOnce also allows inline comments of the form [name: comment] (with
a space after `name:`), e.g., such as [hpl: here I will make some
remarks to the text]. Inline comments can be removed from the output
by a command-line argument (see Section ref{doconce2formats} for an
example).

__Footnotes.__ Adding a footnote[^footnote] is also possible.

[^footnote]: The syntax for footnotes is borrowed from Extended Markdown.

__Tables.__
Tables are also written in the plain text way, 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:

3 First a Section Heading

Section headings have 7 equality characters before and after the heading. With 9 characters we have a chapter heading, 5 gives a subsection heading, and 3 a subsubsection heading.

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. Quotations appear inside double backticks and double single quotes, as in “this example”.

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

Hyperlinks. Paragraph headings are surrounded by double underscores. 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`. DocOnce distinguishes between paper and screen output. In traditional paper output, in PDF generated from LaTeX generated from DocOnce, the URLs of links appear as footnotes. With screen output, all links are clickable hyperlinks, except in the plain text format which does not support hyperlinks.

Cross-references. 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 3.1. References to equations such as (1) work in the same way.

Inline comments. DocOnce also allows inline comments such as **hpl 1**: *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 4 for an example). Ordinary comment lines start with # and are copied to comment lines in the output format.

Footnotes. Adding a footnote[1] is also possible.

Tables. Tables are also written in the plain text way, 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 formats not supporting LaTeX mathematics 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. If you write a lot of mathematics, only the output formats `latex`, `pdflatex`, `html`, `sphinx`, and `pandoc` are of interest and all these support inline LaTeX mathematics so then you will naturally drop the pipe symbol and write just

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

However, if you want more textual formats, like plain text or reStructuredText, the text after the pipe symbol may help to make the math formula more readable if there are backslahes or other special LaTeX symbols in the LaTeX code.

Blocks of mathematics are typeset with raw LaTeX, inside `!bt` and `!et` (begin TeX, end TeX) instructions:

```
!bt
\begin{align}
{\partial u\over\partial t} &= \nabla^2 u + f, label{myeq1}\\
{\partial v\over\partial t} &= \nabla\cdot(q(u)\nabla v) + g
\end{align}
!et
```

[1] The syntax for footnotes is borrowed from Extended Markdown.

The result looks like this:

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

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

Of course, such blocks only looks nice in formats with support for LaTeX mathematics (this includes `latex`, `pdflatex`, `html`, `sphinx`, `ipynb`, `pandoc`, and `mwiki`). Simpler formats have to just list the raw LaTeX syntax.

LaTeX writers who adopt DocOnce need to pay attention to the following:

- AMS LaTeX mathematics is supported, also for the `html`, `sphinx`, and `ipynb` formats.
- Only five equation environments can be used: `\[ ... \]`, `equation*`, `equation`, `align*`, and `align`.
- Newcommands in mathematical formulas are allowed, but not in the running text. Newcommands must be defined in files with names `newcommands*.tex`.
- MediaWiki (`mwiki`) does not support references to equations.

(DocOnce performs extensions to `sphinx` and other formats such that labels in `align` environments work well.)

Remark.

Although DocOnce allows user-defined styles in the preamble of LaTeX output, HTML-based output cannot make use of such styles. If-else constructs for the preprocessor can be used to allow special LaTeX environments for LaTeX output and alternative typesetting for other formats, but it is recommended to stay away from special environments in the text and write in a simpler fashion. For example, DocOnce has no special construction for algorithms, so these must be simulated by lists or verbatim blocks. Other constructions that should be avoided include margin notes, special tables, and `subfigure` (combine image files to one file instead, via `doconce combine_images`).

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

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

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

Such blocks are formatted as

```
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 formats close to plain text), not directly after a section/paragraph heading or a table.

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

Another document can be included by writing `# #include "mynote.do.txt"` at the beginning of a line. 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 plain text editing capabilities.

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

DocOnce supports a type of macros via a LaTeX-style *newcommand* construction. The newcommands are defined in files with names `newcommands*.tex`, using standard LaTeX syntax. Only newcommands for use inside math environments are supported.

Labels, corss-references, citations, and support of an index and bibliography are much inspired by LaTeX syntax, but DocOnce features no backslashes. Use labels for sections and equations only, and preceed the reference by "Section" or "Chapter", or in case of an equation, surround the reference by parenthesis.

Here is an example:

```
===== My Section =====
label{sec:mysec}

idx{key equation} idx{$\u$ conservation}

We refer to Section ref{sec:yoursec} for background material on
the *key equation*. Here we focus on the extension

# \Ddt, \u and \mycommand are defined in newcommands_keep.tex

!bt
\begin{equation}
\Ddt{\u} = \mycommand{v},
label{mysec:eq:Dudt}
\end{equation}
!et
where $\Ddt{\u}$ is the material derivative of $\u$.
Equation (ref{mysec:eq:Dudt}) is important in a number
of contexts, see cite{Larsen_et_al_2002,Johnson_Friedman_2010a}.
Also, cite{Miller_2000} supports such a view.
```

```
As see in Figure ref{mysec:fig:myfig}, the key equation
features large, smooth regions *and* abrupt changes.

FIGURE: [fig/myfile, width=600 frac=0.9] My figure. label{mysec:fig:myfig}

===== References =====

BIBFILE: papers.pub
```

For further details on functionality and syntax we refer to the `doc/manual/manual.do.txt` file in the DocOnce source and a Sphinx version of this document.

4 From DocOnce to Other Formats

Transformation of a DocOnce document `mydoc.do.txt` to various other formats is done with the script `doconce format`:

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

or just drop the extension:

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

4.1 Generating a makefile

Producing HTML, Sphinx, and in particular LaTeX documents from DocOnce sources requires a few commands. Often you want to produce several different formats. The relevant commands should then be placed in a script that acts as a "makefile".

The `doconce makefile` can be used to automatically generate such a makefile, more precisely a Python script `make.py`, which carries out the commands explained below. If our DocOnce source is in `main_myproj.do.txt`, we run

```
doconce makefile main_myproj html pdflatex sphinx
```

to produce the necessary output for generating HTML, PDFLaTeX, and Sphinx. Usually, you need to edit `make.py` to really fit your needs. Some examples lines are inserted as comments to show various options that can be added to the basic commands. A handy feature of the generated `make.py` script is that it inserts checks for successful runs of the many `doconce` commands, and if something goes wrong, the script aborts.

4.2 Preprocessing

The `preprocess` and `mako` programs are used to preprocess the file. The DocOnce program detects whether `preprocess` and/or `mako` statements are present and runs the corresponding programs, first `preprocess` and then `mako`.

Variables to `preprocess` and/or `mako` can be added after the filename with the syntax `-DMYVAR`, `-DMYVAR=val` or `MYVAR=val`.

- The form `-DMYVAR` defines the variable `MYVAR` for `preprocess` (like the same syntax for the C preprocessor - `MYVAR` is defined, but has not specific value). When running `mako`, `-DMYVAR` means that `MYVAR` has the (Python) value `True`.
- The expressions `-DMYVAR=val` and `MYVAR=val` are equivalent. When running `preprocess`, `MYVAR` is defined and has the value `val` (`# #ifdef MYVAR` and `# #if MYVAR == "val"` are both true tests), while for `mako`, `MYVAR` exists as variable and has the value `val` (`% if MYVAR == "val"` is true).

Note that `MYVAR=False` defines `MYVAR` in `preprocess` and any test `# #ifdef MYVAR` is always true, regardless of the value one has set `MYVAR` to, so a better test is `# #if MYVAR == True`. In general, it is recommended to go with `preprocess` directives if the tests are very simple, as in `# #ifdef MYVAR` or `# #if FORMAT == "latex"`, otherwise use only `mako` syntax like `% if MYVAR or YOURVAR:` to incorporate `if` tests in the preprocessor phases.

Two examples on defining preprocessor variables are

```
Terminal> doconce format sphinx mydoc -Dextra_sections -DVAR1=5
Terminal> doconce format sphinx mydoc extra_sections=True VAR1=5
```

The variable `FORMAT` is always defined as the current format when running `preprocess` or `mako`. That is, in the above examples, `FORMAT` is defined as `sphinx`. Inside the DocOnce document one can then perform format specific actions through tests like `#if FORMAT == "sphinx"` (for `preprocess`) or `% if FORMAT == "sphinx":` (for `mako`).

The result of running `preprocess` on a DocOnce file `mydoc.do.txt` is available in a file `tmp_preprocess__mydoc.do.txt`. Similarly, the result of running `mako` is available in `tmp_mako__mydoc.do.txt`. By examining these files one can see exactly what the preprocessors have done.

4.3 Removal of inline comments

The command-line arguments `--no_preprocess` and `--no_mako` turn off running `preprocess` and `mako`, respectively.

Inline comments in the text are removed from the output by

```
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.

4.4 Notes

DocOnce does not have a tag for longer notes, because implementation of a "notes feature" is so easy using the `preprocess` or `mako` programs. Just introduce some variable, say `NOTES`, that you define through `-DNOTES` (or not) when running `doconce format ...`. Inside the document you place your notes between `# #ifdef NOTES` and `# #endif` preprocess tags. Alternatively you use `% if NOTES:` and `% endif` that `mako` will recognize. In the same way you may encapsulate unfinished material, extra material to be removed for readers but still nice to archive as part of the document for future revisions.

4.5 Demo of different formats

A simple scientific report is available in a lot of different formats. How to create the different formats is explained in more depth in the coming sections.

4.6 HTML

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

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

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

Typesetting of Code. If the Pygments package (including the `pygmentize` program) is installed, code blocks are typeset with aid of this package. The command-line argument `--no_pygments_html` turns off the use of Pygments and makes code blocks appear with plain (`pre`) HTML tags. The option `--pygments_html_linenos` turns on line numbers in Pygments-formatted code blocks. A specific Pygments style is set by `--pygments_html_style=style`, where `style` can be `default`, `emacs`, `perldoc`, and other valid names for Pygments styles.

Handling of Movies. MP4, WebM, and Ogg movies are typeset with the HTML5 `video` tag and the HTML code tries to load as many versions among MP4, WebM, and Ogg as exist (and the files are loaded in the mentioned order). If just the specified file is to be loaded, use the `--no_mp4_webm_ogg_alternatives` command-line option. Other movie formats, e.g., `.flv`, `.mpeg` and `.avi`, are embedded via the older `embed` tag.

HTML Styles. The HTML style can be defined either in the header of the HTML file, using a named built-in style; in an external CSS file; or in a template file.

An external CSS file `filename` used by setting the command-line argument `-css=filename`. There available built-in styles are specified as `--html_style=name`, where `name` can be

- `solarized`: the famous solarized style (yellowish),
- `blueish`: a simple style with blue headings (default),
- `blueish2`: a variant of *bluish*,
- `bloodish`: as `bluish`, but dark read as color.

Using `-css=filename` where `filename` is a non-existing file makes DocOnce write the built-in style to that file. Otherwise the HTML links to the CSS stylesheet in `filename`. Several stylesheets can be specified: `-ccs=file1.css,file2.css,file3.css`.

HTML templates. Templates are HTML files with "slots" `%(main)s` for the main body of text, `%(title)s` for the title, and `%(date)s` for the date. DocOnce comes with a few templates. The usage of templates is described in a "separate document": "". That document describes how you your DocOnce-generated HTML file can have any specified layout.

The HTML file can be embedded in a template with your own tailored design, see a tutorial on this topic. The template file must contain valid HTML code and can have three "slots": `%(title)s` for a title, `%(date)s` for a date, and `%(main)s` for the main body of text. The latter is the DocOnce document translated to HTML. The title becomes the first heading in the DocOnce document, or the title (but a title is not recommended when using templates). The date is extracted from the `DATE:` line. With the template feature one can easily embed the text in the look and feel of a website. DocOnce comes with two templates in `bundled/html_styles`. Just copy the directory containing the template and the CSS and JavaScript files to your document directory, edit the template as needed (also check that paths to the `css` and `js` subdirectories are correct - according to how you store the template files), and run

```
Terminal> doconce format html mydoc --html_template=mytemplate.html
```

The template in `style_vagrant` also needs an extra option `--html_style=vagrant`. With this style, one has nice navigation buttons that are used if the document contains `!split` commands for splitting it into many pages.

The HTML File Collection. There are usually a range of files needed for an HTML document arising from a DocOnce source. The needed files are listed in `.basename_html_file_collection`, where `basename` is the filestem of the DocOnce file (i.e., the DocOnce source is in `basename.do.txt`).

Filenames. An HTML version of a DocOnce document is often made in different styles, calling for a need to rename the HTML output file. This is conveniently done by the `--html_output=basename` option, where `basename` is the filestem of the associated HTML files. The `.basename_html_file_collection` file lists all the needed files for the HTML document. Here is an example on making three versions of the HTML document: `mydoc_bloodish.html`, `mydoc_solarized`, and `mydoc_vagrant`.

```
Terminal> doconce format html mydoc --html_style=bloodish \
          --html_output=mydoc_bloodish
Terminal> doconce split_html mydoc_bloodish.html
Terminal> doconce format html mydoc --html_style=solarized \
          --html_output=mydoc_solarized \
          --pygments_html_style=perldoc --html_admon=apricot
Terminal> doconce format html mydoc --html_style=vagrant \
          --html_output=mydoc_vagrant --pygments_html_style=default \
          --html_template=templates/my_adapted_vagrant_template.html
Terminal> doconce split_html mydoc_vagrant.html
```

4.7 Blog Posts

DocOnce can be used for writing blog posts provided the blog site accepts raw HTML code. Google's Blogger service (`blogger.com` or `blogname.blogspot.com`) is particularly well suited since it also allows extensive LaTeX mathematics via MathJax.

1. Write the text of the blog post as a DocOnce document without any title, author, and date.
2. Generate HTML as described above.
3. Copy the text and paste it into the text area in the blog post (just delete the HTML code that initially pops up in the text area). Make sure the input format is HTML.

See a simple blog example and a scientific report for demonstrations of blog posts at `blogspot.no`.

Warning.

The comment field after the blog post does not recognize MathJax (LaTeX) mathematics or code with indentation. However, using a MathJax bookmarklet, e.g., at http://checkmyworking.com/misc/mathjax-bookmarklet/, one can get the mathematics properly rendered. The comment fields are not suitable for computer code, though, as HTML tags are not allowed.

Notice.

Figure files must be uploaded to some web site and the local filenames name must be replaced by the relevant URL. This is usually done by using the `--figure_prefix=http://project.github.io/...` option to give some URL as prefix to all figure names (a similar `--movie_prefix=` option exists as well).

Changing figure names in a blog post can also be done "manually" by some editing code in the script that compiles the DocOnce document to HTML format:

```
cp mydoc.do.txt mydoc2.do.txt
url="https//raw.github.com/someuser/someuser.github.com"
dir="master/project/dir1/dir2"
for figname in fig1 fig2 fig3; do
  doconce replace "[$figname," "[$site/$dir/$figname.png," \
          mydoc2.do.txt
done
doconce format html mydoc2
# Paste mydoc2.html into a new blog post page
```

Blog posts at Google can also be published automatically through email. A Python program can send the contents of the HTML file to the blog site's email address using the packages `smtplib` and `email`.

WordPress (`wordpress.com`) allows raw HTML code in blogs, but has very limited LaTeX support, basically only formulas. The `-wordpress` option to `doconce` modifies the HTML code such that all equations are typeset in a way that is acceptable to WordPress. Look at a simple doconce example and a scientific report to see blog posts with mathematics and code on WordPress.

Speaking of WordPress, the related project http://pressbooks.com can take raw HTML code (from DocOnce, for instance) and produce very nice-looking books. There is no support for mathematics in the text, though.

4.8 Pandoc and Markdown

Output in Pandoc's extended Markdown format results from

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

The name of the output file is `mydoc.md`. From this format one can go to numerous other formats:

```
Terminal> pandoc -R -t mediawiki -o mydoc.mwk --toc mydoc.md
```

Pandoc supports `latex`, `html`, `odt` (OpenOffice), `docx` (Microsoft Word), `rtf`, `texinfo`, to mention some. The `-R` option makes Pandoc pass raw HTML or LaTeX to the output format instead of ignoring it, while the `-toc` option generates a table of contents. See the Pandoc documentation for the many features of the `pandoc` program.

Markdown to HTML conversion. The HTML output from `pandoc` needs adjustments to provide full support for MathJax LaTeX mathematics, and for this purpose one should use `doconce md2html`:

```
Terminal> doconce format pandoc mydoc
Terminal> doconce m2html mydoc
```

The result `mydoc.html` can be viewed in a browser.

Using Pandoc to go from LaTeX to MS Word or HTML. Pandoc is useful to go from LaTeX mathematics to, e.g., HTML or MS Word. There are two ways (experiment to find the best one for your document): `doconce format pandoc` and then translating using `doconce md2latex` (which runs `pandoc`), or `doconce format latex`, and then going from LaTeX to the desired format using `pandoc`. Here is an example on the latter strategy:

```
Terminal> doconce format latex mydoc
Terminal> doconce ptex2tex mydoc
Terminal> doconce replace '\Verb!' '\verb!' mydoc.tex
Terminal> pandoc -f latex -t docx -o mydoc.docx mydoc.tex
```

When we go through `pandoc`, only single equations, `align`, or `align*` environments are well understood for output to HTML.

Note that DocOnce applies the `Verb` macro from the `fancyvrb` package while `pandoc` only supports the standard `verb` construction for inline verbatim text. Moreover, quite some additional `doconce replace` and `doconce subst` edits might be needed on the `.mkd` or `.tex` files to successfully have mathematics that is well translated to MS Word. Also when going to reStructuredText using Pandoc, it can be advantageous to go via LaTeX.

Strict Markdown. The option `--strict_markdown_output` generates plain or strict Markdown without the many extension that Pandoc accepts in Markdown syntax.

GitHub-flavored Markdown. Adding the command-line option `github-md` turns on the GutHub-flavored Markdown dialect, which is used for the issue tracker on GitHub. A special feature is the support of task lists: unnumbered lists with `[x]` (task done) or `[ ]` (task not done). (Tables get typeset directly as HTML and the syntax for code highlighting is different from Pandoc extended Markdown.) Below is a typical response in a GitHub issue tracker where one first quotes the issue and then provides an answer:

```
!bquote
===== Problems with a function =====

There is a problem with the ‘f(x)‘ function

!bc pycod
def f(x):
    return 1 + x
!ec
This function should be quadratic.
!equote

OK, this is fixed:

!bc pycod
def f(x, a=1, b=1, c=1):
    return a*x**2 + b*x + c
!ec

===== Updated task list =====

   * [x] Offer an ‘f(x)‘ function
   * [ ] Extension to cubic functions
   * [x] Allowing general coefficient in the quadratic function

=== Remaining functionality ===

|------------------------------------------------------------|
| function | purpose                          | state |
|----l-----------l------------------------------l--------------|
|  ‘g(x)‘  | Compute the Gaussian function. | Formula ready. |
|  ‘h(x)‘  | Heaviside function.            | Formula ready. |
|  ‘I(x)‘  | Indicator function.            | Nothing done yet. |
|------------------------------------------------------------|
```

Say this text is stored in a file `mycomments.do.txt`. Running

```
Terminal> doconce format pandoc mycomments --github_md
```

produces the Markdown file `mycomments.md`, which can be pasted into the Write field of the GitHub issue tracker. Turning on Preview shows the typesetting of the quote, compute code, inline verbatim, headings, the task list, and the table.

MultiMarkdown. The option `--multimarkdown_output` generates the MultiMarkdown version of Markdown (as opposed to Pandoc-extended Markdown (default), strict Markdown, or GitHub-flavored Markdown).

Strapdown rendering of Markdown text. Strapdown is a tool that can render Markdown text nicely in a web browser by just inserting an HTML header and footer in the Markdown file and load the file into a browser. The option `-strapdown` outputs the relevant header and footer. The output file must be renamed such that it gets the extension `.html`:

```
Terminal> doconce format pandoc mydoc --strict_markdown_output \
          --strapdown --bootstrap_bootwatch_theme=slate
Terminal> mv mydoc.md mydoc.html
```

The `--bootstrap_bootwatch_theme=theme` option is used to choose a Bootswatch theme whose names are found on the Strapdown page.

4.9 LaTeX

Notice.

XeLaTeX and PDFLaTeX are used very much in the same way as standard LaTeX. The minor differences are described in separate sections of the documentation of the DocOnce to LaTeX translation.

Making a LaTeX file `mydoc.tex` from `mydoc.do.txt` is done in two steps: 1) compile the DocOnce source to the `ptex2tex` format, and 2) compile the `ptex2tex` format to standard LaTeX. The `ptex2tex` format can be viewed as an extended LaTeX. For DocOnce users, the `ptex2tex` format essentially means that the file consists of

1. `if-else` statements for the `preprocess` processor such that LaTeX constructions can be activated or deactivated, and
2. all code environments can be typeset according to a `.ptex2tex.cfg` configuration file.

Point 2 is only of interest if you aim to use a special computer code formatting that requires you to use a configuration file and the `ptex2tex` program.

The reason for generating `ptex2tex` and not standard LaTeX directly from DocOnce was that the `ptex2tex` format shows a range of possible LaTeX constructions for controlling the layout. It can be instructive for LaTeX users to look at this code before choosing specific parts for some desired layout. Experts may also want to edit this code (which should be automated by a script such that the

edits can be repeated when the DocOnce source is modified, see Step 2b below). (Direct control of the LaTeX layout in the `doconce format` program would not spit out alternative LaTeX constructs as is now done through the `ptex2tex` step.)

Going from `ptex2tex` format to standard LaTeX format is enabled by either the `ptex2tex` program or DocOnce's (simplified) version of it: `doconce ptex2tex`. Details are given below.

Information on typesetting of inline verbatim.

The `ptex2tex` and the `doconce ptex2tex` programs take inline verbatim code, typeset with backticks in DocOnce, and translate this to

```
\Verb!text!
```

Thereafter, if `text` does not contain illegal characters for the `\texttt` command, the latter is used instead since then LaTeX can insert linebreaks in the inline verbatim text and hence avoid overfull hboxes.

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

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

An option `-device=paper` makes some adjustments for documents aimed at being printed. For example, links to web resources are associated with a footnote listing the complete web address (URL). (Very long URLs in footnotes can be shortened using services such as http://goo.gl/, http://tinyurl.com/, and https://bitly.com/.) The default, `-device=screen`, creates a PDF file for reading on a screen where links are just clickable.

There are many additional options (run `doconce format -help` and look for options starting with `-latex` to get a more verbose description):

- `--latex_font=helvetica,palatino`
- `--latex_papersize=a4,a6`
- `--latex_bibstyle=plain`
- `--latex_title_layout=titlepage,std,beamer,doconce_heading`

- `--latex_style=std,Springer_lncse,Springer_llncs,Springer_T2,...`
- `--latex_list_of_exercises=loe,toc,none`
- `--latex_fancy_header`
- `--latex_section_headings=std,blue,strongblue,gray,gray-wide`
- `--latex_colored_table_rows=blue,gray,no`
- `--latex_todonotes`
- `--latex_double_spacing`
- `--latex_preamble=filename`
- `--latex_admon=mdfbox,graybox2,grayicon,yellowicon,paragraph,colors1,colors2`
- `--latex_admon_color=0.34,0.02,0.8`
- `--latex_admon_envir_map=2`
- `--latex_exercise_numbering=absolute,chapter`
- `--latex_movie=media9,href,multimedia,movie15`
- `--latex_movie_controls=on`
- `--latex_external_movie_viewer` (for `movie15` package)
- `-xelatex`

Step 2. Run `ptex2tex` (if you have installed the Python ptex2tex package) to make a standard LaTeX file,

```
Terminal> ptex2tex mydoc
```

In case you do not have `ptex2tex`, you may run the (simplified) version that comes with DocOnce:

```
Terminal> doconce ptex2tex mydoc
```

The `ptex2tex` command can set two preprocessor variables:

- `PREAMBLE` to turn the LaTeX preamble on or off (i.e., complete document or document to be included elsewhere - and note that the preamble is only included if the document has a title, author, and date)

- `MINTED` for inclusion of the minted package for typesetting of code with the Pygments tool (which requires `latex` or `pdflatex` to be run with the `-shell-escape` option); not used for `doconce ptex2tex` only in the `ptex2tex` program

If you are not satisfied with the generated DocOnce preamble, you can provide your own preamble by adding the command-line option `--latex_preamble=myfile`. In case `myfile` contains a documentclass definition, DocOnce assumes that the file contains the *complete* preamble you want (not that all the packages listed in the default preamble are required and must be present in `myfile`). Otherwise, `myfile` is assumed to contain *additional* LaTeX code to be added to the DocOnce default preamble.

The `ptex2tex` tool makes it possible to easily switch between many different fancy formattings of computer code in 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 sys` for a terminal session, where `sys` is set to a certain environment in `.ptex2tex.cfg` (e.g., `CodeTerminal`). There are about 40 styles to choose from, and you can easily add new ones.

The `doconce ptex2tex` allows specifications of code environments as well. Here is an example:

```
Terminal> doconce ptex2tex mydoc \
          "sys=\begin{quote}\begin{verbatim}@\end{verbatim}\end{quote}" \
          fpro=minted fcod=minted shcod=Verbatim envir=ans:nt
```

Note that `@` must be used to separate the begin and end LaTeX commands, unless only the environment name is given (such as `minted` above, which implies `\begin{minted}{fortran}` and `\end{minted}` as begin and end for blocks inside `!bc fpro` and `!ec`). Specifying `envir=ans:nt` means that all other environments are typeset with the `anslistings.sty` package, e.g., `!bc cppcod` will then result in `\begin{c++}`. A predefined shortcut as in `shcod=Verbatim-0.85` results in denser vertical spacing (baselinestretch 0.85 in LaTeX terminology), and `shcod=Verbatim-indent` implies indentation of the verbatim text. Alternatively, one can provide all desired parameters `\begin{Verbatim}` instruction using the syntax illustrated for the `sys` environments above.

If no environments like `sys`, `fpro`, or the common `envir` are defined on the command line, the plain `\begin{Verbatim}` and `\end{Verbatim}` instructions are used.

Step 2b (optional). Edit the `mydoc.tex` file to your needs. For example, you may want to substitute `section` by `section*` to avoid numbering of sections, you may want to insert linebreaks (and perhaps space) in the title, etc. This can be automatically edited with the aid of the `doconce replace` and `doconce subst` commands. The former works with substituting text directly, while the

latter performs substitutions using regular expressions. You will use `doconce replace` to edit `section{` to `section*{`:

```
Terminal> doconce replace 'section{' 'section*{' mydoc.tex
```

For fixing the line break of a title, you may pick a word in the title, say "Using", and insert a break after than word. With `doconce subst` this is easy employing regular expressions with a group before "Using" and a group after:

```
Terminal> doconce subst 'title\{(.+)Using (.+)\}' \
          'title{\g<1> \\\\ [1.5mm] Using \g<2>' mydoc.tex
```

A lot of tailored fixes to the LaTeX document can be done by an appropriate set of text replacements and regular expression substitutions. You are anyway encourged to make a script for generating PDF from the LaTeX file so the `doconce subst` or `doconce replace` commands can be put inside the script.

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

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

See the next two sections for compilation with XeLaTeX or PDFLaTeX.

If one wishes to use the minted LaTeX package for typesetting code blocks (`Minted_Python`, `Minted_Cpp`, etc., in `ptex2tex` specified through the `*pro` and `*cod` variables in `.ptex2tex.cfg` or `$HOME/.ptex2tex.cfg`), the minted LaTeX package is needed. This package is automatically included by `doconce ptex2tex` if the minted style is used, while you have to include the `-DMINTED` preprocessor option when running the `ptex2tex` program:

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

If the minted style is used, `latex` (or `pdflatex` or `xelatex`) *must* be run with the `-shell-escape` option:

```
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
```

4.10 PDFLaTeX

Running `pdflatex` instead of `latex` follows almost the same steps, but the start is

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

Then `ptex2tex` is run as explained above, and finally

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

4.11 XeLaTeX

XeLaTeX is an alternative to PDFLaTeX and is run in almost the same way, except for the `-xelatex` flag to `doconce format`:

```
Terminal> doconce format pdflatex mydoc --xelatex
Terminal> doconce ptex2tex mydoc
Terminal> xelatex mydoc
```

4.12 From PDF to e-book formats

PDF (as generated from LaTeX above) can be read on most devices today. However, for Kindle and other devices specialized for e-books you need to convert to their format. The Calibre program can produce epub, mobi, and other e-book formats from PDF, see a description.

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

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

We may now produce various other formats:

```
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. However, it is more convenient to use the program `unovonv` to convert between the many formats OpenOffice supports *on the command line*. Run

```
Terminal> unoconv --show
```

to see all the formats that are supported. For example, the following commands take `mydoc.odt` to Microsoft Office Open XML format, classic MS Word format, and PDF:

```
Terminal> unoconv -f ooxml mydoc.odt
Terminal> unoconv -f doc mydoc.odt
Terminal> unoconv -f pdf mydoc.odt
```

Remark about Mathematical Typesetting. At the time of this writing, there is no easy way to go from DocOnce and LaTeX mathematics to reST and further to OpenOffice and the "MS Word world". Mathematics is only fully supported by `latex` as output and to a wide extent also supported by the `sphinx` output format. Some links for going from LaTeX to Word are listed below.

- http://ubuntuforums.org/showthread.php?t=1033441
- http://tug.org/utilities/texconv/textopc.html
- http://nileshbansal.blogspot.com/2007/12/latex-to-openofficeword.html

4.15 Sphinx

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

```
Terminal> doconce sphinx_dir author="authors' names" \
          title="some title" version=1.0 dirname=sphinxdir \
          theme=mytheme mydoc
```

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 file `mydoc.do.txt` 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'`).

One often just runs the simple command

```
Terminal> doconce sphinx_dir mydoc
```

which creates the Sphinx directory `sphinx-rootdir` with relevant files.

The `doconce sphinx_dir` command generates a script `automake_sphinx.py` for compiling the Sphinx document into an HTML document. Run

```
Terminal> python automake_sphinx.py
```

As the output also tells, you can see the Sphinx HTML version of the document by running

```
Terminal> google-chrome sphinx-rootdir/_build/html/index.html
```

or loading the `index.html` file manually into your favorite web browser.

If you cycle through editing the DocOnce file and watching the HTML output, you should observe that `automake_sphinx.py` does not recompile the DocOnce file if the Sphinx `.rst` version already exists. In each edit-and-watch cycle do

```
Terminal> rm mydoc.rst; python automake_sphinx.py
```

Tip.

If you are new to Sphinx and end up producing quite some Sphinx documents, you are encouraged to read the Sphinx documentation and study the `automake_sphinx.py` file. Maybe you want to do things differently.

The following paragraphs describes the many possibilities for steering the Sphinx output.

Links. The `automake_sphinx.py` script copies directories named `fig*` over to the Sphinx directory so that figures are accessible in the Sphinx compilation. It also examines `MOVIE:` and `FIGURE:` commands in the DocOnce file to find other image files and copies these too. I strongly recommend to put files to which there are local links (not `http:` or `file:` URLs) in a directory named `_static`. The `automake_sphinx.py` copies `_static*` to the Sphinx directory, which guarantees that the links to the local files will work in the Sphinx document.

There is a utility `doconce sphinxfix_localURLs` for checking links to local files and moving the files to `_static` and changing the links accordingly. For example, a link to `dir1/dir2/myfile.txt` is changed to `_static/myfile.txt` and `myfile.txt` is copied to `_static`. However, I recommend instead that you manually copy files to `_static` when you want to link to them, or let your script which compiles the DocOnce document do it automatically.

Themes. DocOnce comes with a rich collection of HTML themes for Sphinx documents, much larger than what is found in the standard Sphinx distribution. Additional themes include `agni`, `basicstrap`, `bootstrap`, `cloud`, `fenics`, `fenics_minimal`, `flask`, `haiku`, `impressjs`, `jal`, `pylons`, `redcloud`, `scipy_lectures`, `slim-agogo`, and `vlinux-theme`.

All the themes are packed out in the Sphinx directory, and the `doconce sphinx_dir` insert lots of extra code in the `conf.py` file to enable easy specification and customization of themes. For example, modules are loaded for the additional themes that come with DocOnce, code is inserted to allow customization of the look and feel of themes, etc. The `conf.py` file is a good starting point for fine-tuning your favorite team, and your own `conf.py` file can later be supplied and used when running `doconce sphinx_dir`: simply add the command-line option `conf.py=conf.py`.

A script `make-themes.sh` can make HTML documents with one or more themes. For example, to realize the themes `fenics`, `pyramid`, and `pylon` one writes

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

The resulting directories with HTML documents are `_build/html_fenics` and `_build/html_pyramid`, respectively. Without arguments, `make-themes.sh` makes all available themes (!). With `make-themes.sh` it is easy to check out various themes to find the one that is most attractive for your document.

You may supply your own theme and avoid copying all the themes that come with DocOnce into the Sphinx directory. Just specify `theme_dir=path` on the command line, where `path` is the relative path to the directory containing the Sphinx theme. You must also specify a configure file by `conf.py=path`, where `path` is the relative path to your `conf.py` file.

Example. Say you like the `scipy_lectures` theme, but you want a table of contents to appear *to the right*, much in the same style as in the `default` theme (where the table of contents is to the left). You can then run `doconce sphinx_dir`, invoke a text editor with the `conf.py` file, find the line `html_theme == 'scipy_lectures'`, edit the following `nosidebar` to `false` and `rightsidebar` to `true`. Alternatively, you may write a little script using `doconce replace` to replace a portion of text in `conf.py` by a new one:

```
doconce replace "elif html_theme == 'scipy_lectures':
    html_theme_options = {
        'nosidebar': 'true',
        'rightsidebar': 'false',
        'sidebarbgcolor': '#f2f2f2',
        'sidebartextcolor': '#20435c',
        'sidebarlinkcolor': '#20435c',
        'footerbgcolor': '#000000',
        'relbarbgcolor': '#000000',
    }" "elif html_theme == 'scipy_lectures':
    html_theme_options = {
        'nosidebar': 'false',
        'rightsidebar': 'true',
        'sidebarbgcolor': '#f2f2f2',
        'sidebartextcolor': '#20435c',
        'sidebarlinkcolor': '#20435c',
        'footerbgcolor': '#000000',
        'relbarbgcolor': '#000000',
    }" conf.py
```

Obviously, we could also have changed colors in the edit above. The final alternative is to save the edited `conf.py` file somewhere and reuse it the next time `doconce sphinx_dir` is run

```
doconce sphinx_dir theme=scipy_lectures \
                   conf.py=../some/path/conf.py mydoc
```

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

Step 1. Translate DocOnce into the Sphinx format:

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

Step 2. Create a Sphinx root directory 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
```

The autogenerated `conf.py` file may need some edits if you want to specific layout (Sphinx themes) of HTML pages. The `doconce sphinx_dir` generator makes an extended `conv.py` file where, among other things, several useful Sphinx extensions are included.

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

```
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`. Links to local files in `mydoc.rst` must be modified to links to files in the `_static` directory, see comment above.

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
```

Sphinx can generate a range of different formats: standalone HTML, HTML in separate directories with `index.html` files, a large single HTML file, JSON files, various help files (the qthelp, HTML, and Devhelp projects), epub, LaTeX, PDF (via LaTeX), pure text, man pages, and Texinfo files.

Step 6. View the result:

```
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 LaTeX output.

4.16 Wiki Formats

There are many different wiki formats, but DocOnce only supports three: Googlecode wiki, MediaWiki, and Creole Wiki. These formats are called `gwiki`, `mwiki`, and `cwiki`, respectively. Transformation from DocOnce to these formats is done by

```
Terminal> doconce format gwiki mydoc.do.txt
Terminal> doconce format mwiki mydoc.do.txt
Terminal> doconce format cwiki mydoc.do.txt
```

The produced MediaWiki can be tested in the sandbox of wikibooks.org. The format works well with Wikipedia, Wikibooks, and ShoutWiki, but not always well elsewhere (see this example).

Large MediaWiki documents can be made with the Book creator. From the MediaWiki format one can go to other formats with aid of mwlib. This means

that one can easily use DocOnce to write Wikibooks and publish these in PDF and MediaWiki format, while at the same time, the book can also be published as a standard LaTeX book, a Sphinx web document, or a collection of HTML files.

The Googlecode wiki document, `mydoc.gwiki`, is most conveniently stored in a directory which is a clone of the wiki part of the Googlecode project. This is far easier than copying and pasting the entire text into the wiki editor in a web browser.

When the DocOnce file contains figures, each figure filename must in the `.gwiki` file 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).

4.17 Google Docs

Google Docs are normally made online in the interactive editor. However, you may upload a DocOnce document to Google Docs. This requires transforming the DocOnce document to one of the accepted formats for Google Docs:

- OpenOffice: `doconce format rst` and then run `rst2odt` (or `rst2odt.py`). Upload the `.odt` file, click *Open...* in Google Drive and choose *Google Docs* as viewer.
- MS Word: `doconce format pandoc` and then run `pandoc` to produce a `.docx` file that can be uploaded to Google Drive and opened in Google Docs.
- RTF: `doconce format pandoc` and then run `pandoc` to produce a `.rtf` file that can be uploaded to Google Drive and opened. Another possibility is to run `doconce format latex` and then latex2rtf (the support of mathematics has gotten worse).
- Plain text: `doconce format plain`. Upload the `.txt` file to Google Drive and open in Google Docs.
- HTML: `doconce format html`. Upload the `.html` file and open in Google Docs. Complicated HTML files can be misinterpreted by Google Docs.

This is not yet much tested. It remains to see how code becomes in Google Docs. Support for mathematics is probably impossible until Google Docs can import LaTeX files, but LaTeX mathematics can be embedded in Google Docs and the googledoc2latex script can convert a Google document to LaTeX.

4.18 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.

4.19 Demos

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

```
doc/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.

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.

5 Installation of DocOnce and its Dependencies

Below, we explain the manual installation of all software that may be needed when working with DocOnce documents. The impatient way to install what is needed is to run the `install_doconce.sh` (or `install_doconce.py`) script.

5.1 DocOnce

DocOnce itself is pure Python code hosted at https://github.com/hplgit/doconce. Installation can be done via

```
sudo pip install -e git+https://github.com/hplgit/doconce#egg=doconce
# or if doconce is already installed
sudo pip install -e git+https://github.com/hplgit/doconce#egg=doconce --upgrade
```

Alternatively, one can run the standard procedure

```
git clone git@github.com:hplgit/doconce.git
cd doconce
sudo python setup.py install
cd ..
```

Since DocOnce is frequently updated, it is recommended to use the above procedure and whenever a problem occurs, make sure to update to the most recent version:

```
cd doconce
git pull origin master
sudo python setup.py install
```

5.2 Dependencies

Producing HTML documents, plain text, pandoc-extended Markdown, and wikis can be done without installing any other software. However, if you want other formats as output (LaTeX, Sphinx, reStructuredText) and assisting utilities such as preprocesors, spellcheck, file differences, bibliographies, and so on, the software below must be installed.

To install all needed packages on a GNU/Linux Debian system, such as Ubuntu, you can jump to the script in Section 5.3.

Preprocessors. If you make use of the Preprocess preprocessor, this program must be installed:

```
svn checkout http://preprocess.googlecode.com/svn/trunk/ preprocess
cd preprocess
cd doconce
sudo python setup.py install
cd ..
```

A much more advanced alternative to Preprocess is Mako. Its installation is most conveniently done by `pip`,

```
pip install Mako
```

This command requires `pip` to be installed. On Debian Linux systems, such as Ubuntu, the installation is simply done by

```
sudo apt-get install python-pip
```

Alternatively, one can install from the `pip` source code.

Mako can also be installed directly from source: download the tarball, pack it out, go to the directory and run the usual `sudo python setup.py install`.

Image file handling. Different output formats require different formats of image files. For example, PostScript or Encapuslated PostScript is required for `latex` output, while HTML needs JPEG, GIF, or PNG formats. DocOnce calls up programs from the ImageMagick suite for converting image files to a proper format if needed. The ImageMagick suite can be installed on all major platforms. On Debian Linux (including Ubuntu) systems one can simply write

```
sudo apt-get install imagemagick
```

The convenience program `doconce combine_images`, for combining several images into one, will use `montage` and `convert` from ImageMagick and the `pdftk`, `pdfnup`, and `pdfcrop` programs from the `texlive-extra-utils` Debian package. The latter gets installed by

```
sudo apt-get install texlive-extra-utils
```

Automatic image conversion from EPS to PDF calls up `epstopdf`, which can be installed by

```
sudo apt-get install texlive-font-utils
```

Spellcheck. The utility `doconce spellcheck` applies the `ispell` program for spellcheck. On Debian (including Ubuntu) it is installed by

```
sudo apt-get install ispell
```

Bibliography. The Python package Publish is needed if you use a bibliography in your document. On the website, click on *Clone*, copy the command and run it:

```
hg clone https://bitbucket.org/logg/publish
```

Thereafter go to the `publish` directory and run the `setup.py` script for installing Publish:

```
cd publish
sudo python setup.py
```

Ptex2tex for LaTeX Output. To make LaTeX documents with very flexible choice of typesetting of verbatim code blocks you need ptex2tex, which is installed by

```
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install
```

It may happen that you need additional style files, you can run a script, `cp2texmf.sh`:

```
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../..
```

This script copies some special stylefiles that that `ptex2tex` potentially makes use of. Some more standard stylefiles are also needed. These are installed by

```
sudo apt-get install texlive
```

on Debian Linux (including Ubuntu) systems. TeXShop on Mac comes with the necessary stylefiles (if not, they can be found by googling and installed manually in the `/texmf/tex/latex/misc` directory).

Note that the `doconce ptex2tex` command, which needs no installation beyond DocOnce itself, can be used as a simpler alternative to the `ptex2tex` program.

The *minted* LaTeX style is offered by `ptex2tex` and `doconce ptext2tex` and popular among many users. This style requires the package Pygments to be installed. On Debian Linux,

```
sudo apt-get install python-pygments
```

Alternatively, the package can be installed manually:

```
hg clone ssh://hg@bitbucket.org/birkenfeld/pygments-main pygments
cd pygments
sudo python setup.py install
```

One can also do the simple

```
pip install sphinx
```

which also installs pygments.

If you use the minted style together with `ptex2tex`, you have to enable it by the `-DMINTED` command-line argument to `ptex2tex`. This is not necessary if you run the alternative `doconce ptex2tex` program.

All use of the minted style requires the `-shell-escape` command-line argument when running LaTeX, i.e., `latex -shell-escape` or `pdflatex -shell-escape`.

Inline comments apply the `todonotes` LaTeX package if the option `--latex_todonotes` is given. The `todonotes` package requires several other packages: `xcolor`, `ifthen`, `xkeyval`, `tikz`, `calc`, `graphicx`, and `setspace`. The relevant Debian packages for installing all this are listed below.

LaTeX packages. Many LaTeX packages are potentially needed (depending on various preprocessor variables given to `ptex2tex` or `doconce ptex2tex`. The standard packages always included are `relsize`, `epsfig`, `makeidx`, `setspace`, `color`, `amsmath`, `amsfonts`, `xcolor`, `bm`, `microtype`, `titlesec`, and `hyperref`. The `ptex2tex` package (from ptex2tex) is also included, but removed again if `doconce ptex2tex` is run instead of the `ptex2tex` program, meaning that if you do not use `ptex2tex`, you do not need `ptex2tex.sty`. Optional packages that might be included are `minted`, `fontspec`, `xunicode`, `inputenc`, `helvet`, `mathpazo`, `wrapfig`, `calc`, `ifthen`, `xkeyval`, `tikz`, `graphicx`, `setspace`, `shadow`, `disable`, `todonotes`, `lineno`, `xr`, `framed`, `mdframe`, `movie15`, `a4paper`, and `a6paper`.

Relevant Debian packages that gives you all of these LaTeX packages are

```
texlive
texlive-extra-utils
texlive-latex-extra
texlive-font-utils
```

On old Ubuntu 12.04 one has to do `sudo add-apt-repository ppa:texlive-backports/ppa` and `sudo apt-get update` first, or alternatively install these as well:

```
texlive-math-extra
texlive-bibtex-extra
texlive-xetex
texlive-humanities
texlive-pictures
```

Alternatively, one may pull in `texlive-full` to get all available style files.

If you want to use the *anslistings* code environment with `ptex2tex` (`.ptex2tex.cfg` styles `Python_ANS`, `Python_ANSt`, `Cpp_ANS`, etc.) or `doconce ptex2tex` (`envir=ans` or `envir=ans:nt`), you need the `anslistings.sty` file. It can be obtained from the ptex2tex source. It should get installed by the `cp2texmf.sh` script executed above.

reStructuredText (reST) Output. The `rst` output from DocOnce allows further transformation to LaTeX, HTML, XML, OpenOffice, and so on, through the docutils package. The installation of the most recent version can be done by

```
svn checkout \
  http://docutils.svn.sourceforge.net/svnroot/docutils/trunk/docutils
cd docutils
sudo python setup.py install
cd ..
```

The command

```
pip install sphinx
```

installs Docutils along with Sphinx and Pygments.

To use the OpenOffice suite you will typically on Debian systems install

```
sudo apt-get install unovonv libreoffice libreoffice-dmaths
```

There is a possibility to create PDF files from reST documents using ReportLab instead of LaTeX. The enabling software is rst2pdf. Either download the tarball or clone the svn repository, go to the `rst2pdf` directory and run the usual `sudo python setup.py install`.

Sphinx Output. Output to `sphinx` requires of course the Sphinx software, installed by

```
hg clone https://bitbucket.org/birkenfeld/sphinx
cd sphinx
sudo python setup.py install
cd ..
```

An alternative is

```
pip install sphinx
```

DocOnce comes with many Sphinx themes that are not part of the standard Sphinx source distribution. Some of these themes require additional Python/Sphinx modules to be installed:

- cloud and redcloud: `https://bitbucket.org/ecollins/cloud_sptheme`
- bootstrap: `https://github.com/ryan-roemer/sphinx-bootstrap-theme`
- solarized: `https://bitbucket.org/miiton/sphinxjp.themes.solarized`

- impressjs: https://github.com/shkumagai/sphinxjp.themes.impressjs
- sagecellserver: https://github.com/kriskda/sphinx-sagecell

These must be downloaded or cloned, and `setup.py` must be run as shown above.

Markdown and Pandoc Output. The DocOnce format `pandoc` outputs the document in the Pandoc extended Markdown format, which via the `pandoc` program can be translated to a range of other formats. Installation of Pandoc, written in Haskell, is most easily done by

```
sudo apt-get install pandoc
```

on Debian (Ubuntu) systems.

Epydoc Output. When the output format is `epydoc` one needs that program too, installed by

```
svn co https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc epydoc
cd epydoc
sudo make install
cd ..
```

Remark. Several of the packages above installed from source code are also available in Debian-based system through the `apt-get install` command. However, we recommend installation directly from the version control system repository as there might be important updates and bug fixes. For `svn` directories, go to the directory, run `svn update`, and then `sudo python setup.py install`. For Mercurial (`hg`) directories, go to the directory, run `hg pull; hg update`, and then `sudo python setup.py install`.

Analyzing file differences. The `doconce diff file1 file prog` command for illustrating differences between two files `file1` and `file2` using the program `prog` requires `prog` to be installed. By default, `prog` is `difflib` which comes with Python and is always present if you have DocOnce installed. Another choice, `diff`, should be available on all Unix/Linux systems. Other choices, their URL, and their `sudo apt-get install` command on Debian (Ubuntu) systems appear in the table below.

Program	URL	Debian/Ubuntu install
`pdiff`	a2ps wdiff	`sudo apt-get install a2ps wdiff texlive-latex-extra texlive-latex-re`
`latexdiff`	latexdiff	`sudo apt-get install latexdiff`
`kdiff3`	kdiff3	`sudo apt-get install kdiff3`
`diffuse`	diffuse	`sudo apt-get install diffuse`
`xxdiff`	xxdiff	`sudo apt-get install xxdiff`
`meld`	meld	`sudo apt-get install meld`
`tkdiff.tcl`	tkdiff	not in Debian

5.3 Quick Debian/Ubuntu Install

On Debian (including Ubuntu) systems, it is straightforward to install the long series of DocOnce dependencies:

```
# Version control systems
sudo apt-get install -y mercurial git subversion

# Python
sudo apt-get install -y idle ipython python-pip python-pdftools texinfo

# LaTeX
sudo apt-get install -y texlive texlive-extra-utils texlive-latex-extra texlive-math-extra texlive-font
# or sudo apt-get install -y texlive-full  # get everything
sudo apt-get install -y latexdiff auctex

# Image and movie tools
sudo apt-get install -y imagemagick netpbm mjpegtools pdftk giftrans gv libav-tools libavcodec-extra-53

# Misc
sudo apt-get install -y ispell pandoc libreoffice unoconv libreoffice-dmaths curl a2ps wdiff meld xxdif

# More Python software
sudo pip install sphinx  # install pygments and docutils too
sudo pip install mako
sudo pip install -e svn+http://preprocess.googlecode.com/svn/trunk#egg=preprocess
sudo pip install -e hg+https://bitbucket.org/logg/publish#egg=publish

sudo pip install -e hg+https://bitbucket.org/ecollins/cloud_sptheme#egg=cloud_sptheme
sudo pip install -e git+https://github.com/ryan-roemer/sphinx-bootstrap-theme#egg=sphinx-bootstrap-them
sudo pip install -e hg+https://bitbucket.org/miiton/sphinxjp.themes.solarized#egg=sphinxjp.themes.solar
sudo pip install -e git+https://github.com/shkumagai/sphinxjp.themes.impressjs#egg=sphinxjp.themes.impr
sudo pip install -e git+https://github.com/kriskda/sphinx-sagecell#egg=sphinx-sagecell
sudo pip install -e svn+https://epydoc.svn.sourceforge.net/svnroot/epydoc/trunk/epydoc#egg=epydoc

# DocOnce itself
rm -rf srclib   # put downloaded software in srclib
mkdir srclib
cd srclib
git clone git@github.com:hplgit/doconce.git
cd doconce
sudo python setup.py install -y
cd ../..

# Ptex2tex
cd srclib
svn checkout http://ptex2tex.googlecode.com/svn/trunk/ ptex2tex
cd ptex2tex
sudo python setup.py install -y
cd latex
sh cp2texmf.sh  # copy stylefiles to ~/texmf directory
cd ../../..
```