An overview of Pandoc

Massimiliano Dominici

Abstract

This paper is a short overview of Pandoc, a utility for the conversion of Markdown-formatted texts to many output formats, including LaTeX and HTML.

1 Introduction

Pandoc is software, written in Haskell, whose aim is to facilitate conversion between some lightweight markup languages and the most widespread 'final' document formats.[1] On the program's website [3], Pandoc is described as a sort of 'swiss army knife' for converting between different formats, and in fact it is able to read simple files written in LaTeX or HTML; but it is of lesser use when trying to translate LaTeX documents with non-trivial constructs such as commands defined by the user or by a dedicated package.

Pandoc shows its real utility, in my opinion, when what is needed is to obtain several output formats from a single source, as in the case of a document distributed online (HTML), in print form (PDF via LaTeX) and for viewing on tablets or ebook readers (EPUB). In such cases one may find that writing the document in a rich format (e.g. LaTeX) and converting later to other markup languages often poses significant problems because of the different 'philosophies' that underlie each language. It is advisable, instead, to choose as a starting point a language that is 'neutral' by design. A good candidate for this role is a lightweight markup language, and in particular Markdown, of which Pandoc is an excellent interpreter.

In this article we will briefly discuss the concept of a 'lightweight markup language' with particular reference to Markdown (§ 2), and then we will review Pandoc in more details (§ 3) before drawing our conclusions (§ 5).

2 Lightweight markup languages: Markdown

Before getting to the heart of the matter, it is advisable to say a few words about lightweight markup languages (LML) in general. They are designed with the explicit goal of minimizing the impact of the markup instructions within the document, with a particular emphasis on the *readability* of the text by a *human being*, even when the latter does not know the (few) conventions that the program follows in order to format the document.

These languages are mainly used in two fields: documentation of code (reStructuredText, AsciiDoc, etc.) and management of contents for the web (Markdown, Textile, etc.). In the case of code documentation, the use of an LML is a good choice, because the documentation is interspersed in the code itself, so it should be easy to read by a developer perusing the code; but at the same time it should be able to be converted to presentation formats (PDF and HTML, traditionally, but today many IDEs include some form of visualization for the internal documentation). In the case of web content, the emphasis is placed on the ease of writing for the user. Many content management systems already provide plugins for one or more of those languages and the same is true for static site generators[2] that are usually built around one of them and often provide support for others. The various wiki dialects can be considered another instance of LML.

The actual 'lightness' of an LML depends greatly on its ultimate purpose. In general, an LML conceived for code documentation will be more complex and less readable than one conceived for web content management, which in turn will often not be capable of general semantic markup. A paradigmatic example of this second category is Markdown that, in its original version, stays rigorously close to the minimalistic approach of the first LMLs. The following citation from its author, John Gruber, explains his intentions in designing Markdown:

> Markdown is intended to be as easy-to-read and easy-to-write as is feasible. Readability, however, is emphasized above all else. A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions.[3]

The only output format targeted by the reference implementation of Markdown is HTML; indeed, Markdown also allows raw HTML code. Gruber has

Translation by the author from his original in *ArsTeXnica* #15, April 2013, "Una panoramica su Pandoc", pp. 31–38.

[1] Strictly speaking, LaTeX isn't a 'final' document format in the same way PDF, ODF, DOC, EPUB, etc. are. But, from the point of view of a Pandoc user, LaTeX is a 'final' — or intermediate, at least — product.

[2] Static site generators are a category of programs that build a website in HTML starting from source files written in a different format. The HTML pages are produced beforehand, usually on a local computer, and then loaded on the server. Websites built this way share a great resemblance with old websites written directly in HTML, but unlike those, in the building process it is possible to use templates, share metadata across pages, and create structure and content programmatically. Static site generators constitute an alternative to the more popular dynamic server applications.

[3] [1], `http://daringfireball.net/projects/markdown/syntax#philosophy`. A significant contribution to the design of Markdown was made by Aaron Swartz.

Table 1: Markdown syntax: inline elements.

Element	Markdown	LaTeX	HTML
Links	[link](http://example.net)	\href{link}{% http://example.net}	<a href="http://example.net/"> link</a>
Emphasis	_emphasis_ *emphasis*	\emph{emphasis} \emph{emphasis}	<em>emphasis</em> <em>emphasis</em>
Strong emphasis	__strong__ **strong**	\textbf{strong} \textbf{strong}	<strong>strong</strong> <strong>strong</strong>
Verbatim	‘printf()‘	\verb|printf()|	<code>printf()</code>
Images	![Alt](/path/to/img.jpg)	\includegraphics{img}	<img src="/path/to/img.jpg" alt="Alt" />

Table 2: Markdown syntax: block elements.

Element	Markdown	LaTeX	HTML

Sections	# Title #
	## Title ##
	...
		\section{Title}
		\subsection{Title}
		...
			<h1>Title</h1>
			<h2>Title</h2>
			...

Quotation	> This paragraph
	> will show
	> as quote.
		\begin{quote}
		This paragraph
		will show
		as quote.
		\end{quote}
			<blockquote><p>
			    This paragraph
			    will show
			    as quote.
			  </p></blockquote>

Itemize	* First item * Second item * Third item	\begin{itemize} \item First item \item Second item \item Third item \end{itemize}	<ul> <li>First item</li> <li>Second item</li> <li>Third item</li> </ul>
Enumeration	1. First item 2. Second item 3. Third item	\begin{enumerate} \item First item \item Second item \item Third item \end{enumerate}	<ol> <li>First item</li> <li>Second item</li> <li>Third item</li> </ol>

Verbatim	Text paragraph.
	
	    grep -i '\$' <file
		Text paragraph.
		
		\begin{verbatim}
		grep -i '\$' <file
		\end{verbatim}
			<p>Text paragraph.</p>
			
			<pre><code>
			  grep -i '\$' &lt;file
			</code></pre>

always adhered to these initial premises and has consistently refused to extend the language beyond the original specifications. This stance has caused a proliferation of variants, so that every single implementation constitutes an 'enhanced' version. Famous websites like GitHub, reddit and Stack Overflow, all support their own Markdown flavour; and the same is true for conversion programs like MultiMarkdown or Pandoc itself, which also introduce new output formats. It's not necessary, here, to examine the details of the different flavours; the reader can get an idea of the basic formatting rules from tables 1 and 2. Of course, in the reference implementation there is no LaTeX output, so I have provided the most logical translation. In the following sections we will see how Pandoc works in practice.

3 An overview of Pandoc

As mentioned in the introduction, Pandoc is primarily a Markdown interpreter with several output formats: HTML, LaTeX, ConTeXt, DocBook, ODF, OOXML, other LMLs such as AsciiDoc, reStructuredText and Textile (a complete list can be found in [3]). Pandoc can also convert, with severe restrictions, a source file in LaTeX, HTML, DocBook, Textile or reStructuredText to one of the aforementioned output formats. Moreover it extends the syntax of Markdown, introducing new elements and providing customization for the elements already available in the reference implementation.

3.1 Markdown syntax extensions

Markdown provides, by design, a very limited set of elements. Tables, footnotes, formulas, and bibliographic references have no specific markup in Markdown. The author's intent is that all markup exceeding the limits of the language should be expressed in HTML. Pandoc maintains this approach (and, for LaTeX or ConTeXt output, allows the use of raw TeX code) but makes it unnecessary, since it introduces many extensions, giving the user proper markup for each of the elements mentioned above. In the following paragraphs we'll take a look at these extensions.

Metadata Metadata for title, author and date can be included at the beginning of the file, in a text block, each preceded by the character %, as in the following example.

```
% Title
% First Author; Second Author
% 17/02/2013
```

The content for any of these elements can be omitted, but then the respective line must be left blank (unless it is the last element, i.e. the date).

```
% Title
%
% 17/02/2013

%
% First Author; Second Author
% 17/02/2013

% Title
% First Author; Second Author
```

Since version 1.12 metadata support has been substantially extended. Now Pandoc accepts multiple metadata blocks in YAML format, delimited by a line of three hyphens (---) at the top and a line of three hyphens (---) or three dots (...) at the bottom.[4] This gives the user a high level of flexibility in setting and using variables for templates (see section 3.1).

YAML structures metadata in arrays, thus allowing for a finer granularity. The user may specify in his source file the following code:

```
---
author:
- name: First Author
- affiliation: First Affiliation
- name: Second Author
- affiliation: Second Affiliation
---
```

[4] YAML Ain't Markup Language, http://www.yaml.org/.

and then, in the template

```
$for(author)$
$if(author.name)$
$author.name$
$if(author.affiliation)$ ($author.affiliation$)
$endif$
$else$
$author$
$endif$
$endfor$
```

to get a list of authors with (if present) affiliations.

As we will see in section 3.1, a YAML block can also be used to build a bibliographic database.

Footnotes Since the main purpose of Markdown and its derivatives is readability, the mark and the text of a footnote should usually be split. It is recommended to write the footnote text just below the paragraph containing the mark, but this is not strictly required: the footnotes could be collected at the beginning or at the end of the document, for instance. The mark is an arbitrary label enclosed in the following characters: [^...]. The same label, followed by ':' must precede the footnote text.

When the footnote text is short, it is possible to write it directly inside the text, without the label. The output from all the footnotes is collected at the end of the document, numbered sequentially. Here is an example of the input:

```
Paragraph containing[^longnote] a
footnote too long to be written directly
inside the text.

[^longnote]: A footnote too long to be
written inside the text without causing
confusion.

New paragraph.^[A short note.]
```

Tables Again, syntax for tables is based on considerations of readability of the source. The alignment of the cells composing the table is immediately visible in the alignment of the text with respect to the dashed line that divides the header from the rest of the table; this line must *always* be present, even when the header is void.[5] When the table includes cells with more than one line of text, it is mandatory

[5] In fact, it is the header, if present, or the first line of the table that sets the alignment of the columns. Aligning the remaining cells is not needed, but it is recommended as an aid for the reader.

to enclose it between two dashed lines. In this case the width of each column in the source file is used to compute the width of the equivalent column in the output table. Multicolumn or multirow cells are not supported. The caption can precede or follow the table; it is introduced by the markup ':' (alternatively: `Table:`) and must be divided from the table itself by a blank line:

```
------------------------------------------
        Right    Centered    Left
------------- -------------- -------------
         Text      Text      Text
      aligned    aligned     aligned
        right     center     left

     New cell    New cell    New cell
------------------------------------------

Table: Alignment
```

There's an alternative syntax to specify the alignment of the individual columns: divide columns with the character '|' and use the character ':' in the dashed line below the header to specify, through its placement, the column's alignment, as shown in the following example:

```
Right    | Centered     | Left
--------:|:-----------:|:-------------
Text     | Text         | Text
aligned  | aligned      | aligned
right    | center       | left
         |              |
New cell | New cell     | New cell

: Alignment by ':'
```

In the examples above, the cells cannot contain 'vertical' material (multiple paragraphs, verbatim blocks, lists). 'Grid' tables (see the example below) allow this, at the cost of not being to specify the column alignments.

```
+-------------+-------------+----------------+
| Text        | Lists       | Code           |
+=============+=============+================+
| Paragraph.  | * Item 1    | ~~~            |
|             | * Item 2    | \def\PD{%      |
| Paragraph.  |             |  \emph{Pandoc} |
|             | * Item 3    | ~~~            |
+-------------+-------------+----------------+
| New cell    | New cell    | New cell       |
+-------------+-------------+----------------+
```

Figures As shown in table 1, Markdown allows for the use of inline images with the following syntax:

`![Alternative text](/path/image)`

where '`Alternative text`' is the description that HTML uses when the image cannot be viewed. Pandoc adds to that one more feature: if the image is divided by blank lines from the remaining text, it will be interpreted as a floating object with its own caption taken from the '`Alternative text`'.

Listings In standard Markdown, verbatim text is marked by being indented by four spaces or one tab. To that, Pandoc adds the ability to specify identifiers, classes and attributes for a given block of 'verbatim' material. Pandoc will treat them in different ways, depending on the output format and the command line options; in some circumstances, they will simply be ignored. To achieve this, Pandoc introduces an alternative syntax for listings of code: instead of indented blocks, they are represented by blocks delimited by sequences of three or more tildes (`~~~`) or backticks (```` ``` ````); identifiers, classes and attributes must follow that initial 'rule', enclosed in braces. In the following example[6] we can see a listing of Python code with, in this order: an identifier, the class that marks it as Python code, another class that specifies line numbering and an attribute that marks the starting point of the numbering.

```
~~~ {#bank .python .numberLines startFrom="5"}
class BankAccount(object):
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def overdrawn(self):
        return self.balance < 0
my_account = BankAccount(15)
my_account.withdraw(5)
print my_account.balance
~~~
```

It is possible to use identifiers, class and attributes for inline code, too:

```
The return value of the `printf`{.C} function
is of type `int`.
```

By default, Pandoc uses a simple `verbatim` environment for code that doesn't need highlighting and the `Highlighting` environment, defined in the template's preamble (see 3.1) and based on `Verbatim` from fancyvrb, when highlighting is needed. If the option `--listings` is given on the command line, Pandoc uses the `lstlistings` environment from listings every time a code block is encountered.

Formulas Pandoc supports mathematical formulas quite well, using the usual TEX syntax. Expressions enclosed in dollar signs will be interpreted as inline formulas; expressions in double dollar signs

[6] From `http://wiki.python.org/moin/SimplePrograms`.

will be interpreted as displayed formulas. This is all comfortably familiar to a TEX user.

The way these expressions will be treated depends on the output format. For TEX (LATEX/ConTEXt) output, the expressions are passed without modifications, except for the substitution of the delimiters for display math: `\[...\]` instead of `$$...$$`. When HTML (or similar) output is required, the behavior is controlled by command line options. Without options, Pandoc will try to render the formulas by means of Unicode characters. Other options allow for the use of some of the most common JavaScript libraries for visualizing math on the web: MathJax, LaTeXMathML and jsMath. It is also possible, always by means of a command line switch, to render formulas as images or to encode them as MathML.[7]

Pandoc is also able to parse simple macros and expand them in output formats different from the supported TEX dialects. This feature, though, is only available in the context of math rendering.

Citations Pandoc can build a bibliography (and manage citations inside the text) using a database in any of several common formats (BIBTEX, EndNote, ISI, etc.). The database file must *always* be specified as the argument of the option `--bibliography`. Without other options on the command line, Pandoc will include citations and bibliographic entries as plain text, formatted following the bibliographic style 'Chicago author–date'. The user may specify a different style by means of the option `--csl`, whose argument is the name of a CSL style file[8] and may also specify that the bibliographic apparatus will be managed by natbib or biblatex. In this case Pandoc will not include in the LATEX output citations and entries in extended form, but only the required commands. The options to get this behavior are, respectively, `--natbib` and `--biblatex`.

The user must type citations in the form `[@key1; @key2;...]` or `@key1` if the citation should not be enclosed in round brackets. A dash preceding the label suppresses the author's name (when supported by the citation format). Bibliographic references are always placed at the end of the document.

[7] The web pages for these different rendering engines for math on the web are `http://www.mathjax.org`, `http://math.etsu.edu/LaTeXMathML`, `http://www.math.union.edu/~dpvc/jsmath` and `http://www.w3.org/Math`.

[8] CSL (`http://citationstyles.org`), *Citation Style Language*, is an open format, XML-based, language to describe the formatting of citations and bibliographies. It is used in several citation managers, such as Zotero, Mendeley, and Papers. A detailed list of the available styles can be found in `http://zotero.org/styles`.

```
---
references:
- author:
    family: Gruber
    given:
    - John
  id: gruber13:_markd
  issued:
    year: 2013
  title: Markdown
  type: no-type
  publisher: <http://daringfireball.net/
              projects/markdown/>
- volume: 32
  page: 272-277
  container-title: TUGboat
  author:
    family: Kielhorn
    given:
    - Axel
  id: kielhorn11:_multi
  issued:
    year: 2011
  title: Multi-target publishing
  type: article-journal
  issue: 3
...
```

Figure 1: A YAML bibliographic database (line break in the url is editorial).

Since version 1.12 native support for citations has been split from the core functions of Pandoc. In order to activate this feature, one must now use an external filter (`--filter pandoc-citeproc`, to be installed separately).[9]

A new feature is that bibliographic databases can now be built using the `references` field inside a YAML block. Finding the correct encoding for a YAML bibliographic database can be a little tricky, so it is recommended, if possible, to convert from an existing database in one of the formats recognized by Pandoc (among them BIBTEX), using the `biblio2yaml` utility, provided together with the `pandoc-citeproc` filter. The YAML code for the first two items in this article's bibliography, created by converting the `.bib` file, is shown in figure 1.

A YAML field can be used also for specifying the CSL style for citations (`csl` field), or the external bibliography file, if required (`bibliography` field).

Raw code (HTML or TEX) All implementations of Markdown, of whichever flavour, allow for the use of raw HTML code, written without modifications in the output, as we mentioned in section 2. Pandoc extends this feature, allowing for the use of TEX raw code, too. Of course, this works only for LATEX/ConTEXt output.

[9] The filter is not needed when using natbib or biblatex directly instead of the native support.

```
  1  \documentclass$if(fontsize)$[$fontsize$]$endif$
  2    {article}
  3  \usepackage{amssymb,amsmath}
  4  \usepackage{ifxetex}
  5  \ifxetex
  6    \usepackage{fontspec,xltxtra,xunicode}
  7    \defaultfontfeatures{Mapping=tex-text,
  8                         Scale=MatchLowercase}
  9  \else
 10     \usepackage[utf8]{inputenc}
 11  \fi
 12  $if(natbib)$
 13  \usepackage{natbib}
 14  \bibliographystyle{plainnat}
 15  $endif$
 16  $if(biblatex)$
 17  \usepackage{biblatex}
 18  $if(biblio-files)$
 19  \bibliography{$biblio-files$}
 20  $endif$
 21  $endif$

     ...

113  $body$
114
115  $if(natbib)$
116  $if(biblio-files)$
117  $if(biblio-title)$
118  $if(book-class)$
119  \renewcommand\bibname{$biblio-title$}
120  $else$
121  \renewcommand\refname{$biblio-title$}
122  $endif$
123  $endif$
124  \bibliography{$biblio-files$}
125  $endif$
126  $endif$
127  $if(biblatex)$
128  \printbibliography
129    $if(biblio-title)$[title=$biblio-title$]$endif$
130  $endif$
131  $for(include-after)$
132  $include-after$
133  $endfor$
134  \end{document}
```

Figure 2: Fragments of the default Pandoc v1.11 template for LaTeX.

Templates One of the most interesting features of Pandoc is the use of customized templates for the different output formats. For HTML-derived and TeX-derived output formats this can be achieved in two ways. First of all, the user may generate only the document 'body' and then include it inside a 'master' (for TeX output, with `\input` or `\include`). In this way, an *ad hoc* preamble can be built beforehand. This is in fact the default behaviour for Pandoc; to obtain a complete document, including a preamble, the command line option `--standalone` (or its equivalent `-s`) is used.

It is also possible to build more flexible templates, useful for different projects with different features, providing for a moderate level of customization. As the reader can see in figure 2, a template is substantially a file in the desired output format (in this case LaTeX) interspersed with variables and control flow statements introduced by a dollar sign. The expressions will be evaluated during the compilation and replaced by the resulting text. For instance, at line 113 of the listing in figure 2 we find the expression `$body$`, which will be replaced by the document body. Above, at lines 12–21, we can find the sequence of commands that will include in the final output all the resources needed to generate a bibliography by means of natbib or biblatex. This code will be activated only if the user has specified either `--natbib` or `--biblatex` on the command line. The code to print the bibliography can be found at the end of the listing, at lines 124–130.

In this way it is possible to define all the desired variables and the respective compiler options. The user can thus change the default template to specify, e.g., among the options that may be passed to the class, not only the body font, but a generic string containing more options.[10] We would replace the first line in the listing of figure 2 with the following:

```
\documentclass$if(clsopts)$[$clsopts$]$endif$
```

Then we can compile with the following options:

```
pandoc -s -t latex --template=mydefault \
  -V clsopts=a4paper,12pt -o test.tex test.md
```

given that we saved the modified template in the current directory by the name `mydefault.latex`.

Since version 1.12, 'variables' can be replaced by YAML 'metadata', specified either inside the source file or on the command line using the `-M` option.

4 Problems and limitations

We've seen many nice features of Pandoc. Not surprisingly, Pandoc also has some limitations and shortcomings. Some of these shortcomings are tied to the particular LML used by Pandoc. For instance, Markdown doesn't allow semantic markup.[11] This kind of limitation can be addressed using an additional level

[10] This can be also be achieved via the variable `$fontsize`. (Since Pandoc 1.12, the default LaTeX template includes separate variables for body font size, paper size and language, and a generic `$classoption` variable for other parameters.)

[11] This is not an issue necessarily pertinent to all LMLs since some of them provide methods to define objects that behave like LaTeX macros, either through pre- or post-processing (`txt2tags`) or by taking advantage of conceptually close structures (the 'class' of a `span` in HTML, in Textile). In any case,

of abstraction, using preprocessors like `gpp` or `m4`, as illustrated by Aditya Mahajan in [4]. Of course, this clashes with the initial purpose of readability and introduces further complexity, though the use of `m4` need not significantly increase the amount of extra markup.

Other problems, though, unexpectedly arise in the process of conversion to LaTeX output. For instance, the cross reference mechanism is calibrated to HTML and shows all its shortcomings with regard to the LaTeX output. The cross reference is in fact generated by means of a hypertext anchor and not by the normal use of `\label` and `\ref`. As a typical example, let's consider a labelled section referenced later in the text, as in the following:

```
## Basic elements ## {#basic}
[...]
As we have explained in
[Basic elements](#basic)
```

we get this result:

```
\hyperdef{}{basic}{%
  \subsection{Basic elements}\label{basic}
}
[...]
As we have explained in
\hyperref[basic]{Basic elements}
```

which is not exactly what a LaTeX user would expect ... Of course one could directly use `\label` and `\ref`, but they will be ignored in all non-TeX output formats. Or, we could use a preprocessor to get two different intermediate source files, one for HTML and for LaTeX (and maybe a third for ODF/OOXML, etc.), but by now the original point of using Pandoc is being lost.

Formulas, too, may cause some problems. Pandoc recognizes only inline and display expressions. The latter are always translated as `displaymath` environments. It is not possible to specify a different kind of environment (`equation`, `gather`, etc.) unless one of the workarounds discussed above is employed, with all the consequent drawbacks also noted.

It should be stressed, however, that Pandoc is a program in active development and that several features present in the current version were not available a short time ago. So it is certainly possible that all or some of the shortcomings that a LaTeX user finds in the current version of Pandoc will be addressed in the near or mid-term. It is also possible, to some extent, to extend or modify Pandoc's behaviour by means of scripts, as noted at `http://johnmacfarlane.net/pandoc/scripting.html`. One major drawback, until recently, was the mandatory use of Haskell for such scripts (a drawback for me, at least ...). The current version also allows Python, thus making easier the task of creating such scripts.[12]

though, the philosophy behind LMLs doesn't support such markup methods.

5 Conclusion

To conclude this overview, I consider Pandoc to be the best choice for a project requiring multiple output formats. The use of a 'neutral' language in the source file makes it easier to avoid the quirks of a specific language and the related problems of translation to other languages. For a LaTeX user in particular, being able to type mathematical expressions 'as in LaTeX' and to use a BibTeX database for bibliographic references are also two strong points.

One should not expect to find in Pandoc an easy solution for every difficulty. Limitations of LMLs in general, and some flaws specific to the program, entail the need for workarounds, making the process less immediate. This doesn't change the fact that, if the user is aware of such limitations and the project can bear them, Pandoc makes obtaining multiple output formats from a single source extremely easy.

References

[1] John Gruber. Markdown. `http://daringfireball.net/projects/markdown/`, 2013.

[2] Axel Kielhorn. Multi-target publishing. *TUGboat*, 32(3):272–277, 2011. `http://tug.org/TUGboat/tb32-3/tb102kielhorn.pdf`.

[3] John MacFarlane. *Pandoc*: a universal document converter. `http://johnmacfarlane.net/pandoc/`, 2013.

[4] Aditya Mahajan. How I stopped worrying and started using Markdown like TeX. `http://randomdeterminism.wordpress.com/2012/06/01/how-i-stopped-worring-and-started-using-markdown-like-tex/`, 2012.

[5] Wikipedia. Lightweight markup language. `http://en.wikipedia.org/wiki/Lightweight_markup_language`, 2013.

◇ Massimiliano Dominici
Pisa, Italy
`mlgdominici (at) gmail dot com`

[12] Another option is writing one's own custom 'writer' in Lua. A writer is essentially a program that translates the data structure, collected by the 'reader', in the format specified by the user. Having Lua installed on the system is not required, since a Lua interpreter is embedded in Pandoc. See `http://johnmacfarlane.net/pandoc/README.html#custom-writers`.