

Markdown to SILE conversion with Pandoc

First version, June 2022.

The source of this document is written in Markdown. A PDF may be generated with the following instructions, provided “Omikhleia’s classes & packages for SILE” are installed¹.

```
pandoc -t pandoc/omimarkdown.lua examples/markdown-sample.md >  
examples/markdown-sample.sil; sile examples/markdown-sample.sil
```

1. Introducing the challenge

Markdown processing with SILE has been, at least, an open question since 2016, if not even earlier².

At least three routes for converting Markdown to SILE can be considered, and are summarized below. All these approaches have legitimate *pros* and *cons*. Be aware that the views presented here are somewhat opinionated!

1.1. Developing a native converter

The “native” way would be to interpret Markdown from SILE itself.

This is what the SILE **markdown** class is trying to achieve, based on the Lua `lunamark` module for parsing Markdown. As of March 2022, the results are still very unconvincing³. It has lots of bugs and is very incomplete.

The very idea of using SILE and Lua to process Markdown is not unsound. But besides its highly broken *proof of concept* status, the fact that this attempt comes as a *class* is also questionable. Class designers can implement lots of cool classes for books, articles, etc., but what does it tell about including Markdown *content* in a document based on these?

Would it be done another way, the *pros* are that SILE would have its own logic, open for customization, to process Markdown without the need of a third party solution. The *cons* are that it requires a good Markdown processing library, that has to be maintained, with a tight integration in SILE.

¹ <https://github.com/Omikhleia/omikhleia-sile-packages>

² <https://github.com/sile-typesetter/sile/issues/413>

³ <https://github.com/sile-typesetter/sile/issues/1336>

1.2. Developing a Pandoc-based converter

Here, we refer to the standard Pandoc use, with that software providing a dedicated writer, likely written in Haskell and integrated in the software, as it does for other output formats.

Some efforts have been done in that direction. It requires a Pandoc fork⁴, which is not yet (as of June 2022) merged upstream. It also requires the SILE **pandoc** package to be used, which has still plenty of TODO comments...

Again, the idea is not unsound. The apparent choice made in the current proposal, though, is to (more or less) expose the Pandoc AST to SILE, which would then implement the necessary actions. This author does believe this approach to be kind of flawed — There could have been ways for the output to look more like it was written with the available tools and packages, rather than in weird Pandoc low-level constructs that might change as the software evolves. Anyhow, would the Pandoc fork be merged into the product, without any other effort in SILE itself, the result could be fairly deceptive too, at this date, for reasons explained later.

Notwithstanding the conversion output format, the *cons*, in my humble opinion, are that it puts the burden of maintaining it on both the Pandoc and the SILE teams. The Pandoc AST is somewhat stable at this date, but changes are likely to be expected anyway. Relying on Pandoc’s releases also means that getting things done is likely to take more time.

1.3. Developing a Pandoc “custom writer” in Lua

Pandoc, in the meantime, has evolved and it now supports creating custom “Writers” in Lua⁵.

While such custom writers may have some rough edges, the idea is quite appealing. After all, SILE is mostly written in Lua, so the skills are there in the community. It has not been considered until a recent date, though. As can be guessed from our introductory note, it is the purpose of the present document to experiment and document that third “route”.

The *pros* are that it is written in the same language as SILE and that it offers an interesting alternative which doesn’t imply modifications of Pandoc or a special build. It also allows us to move fast, somewhat independently from Pandoc. The

⁴ <https://github.com/alquerque/pandoc/commits/sile4>

⁵ <https://pandoc.org/custom-writers.html>

cons is that this custom writer API is fairly recent and might change. Actually, besides what is called the “Classic style”, there’s now even a “New style”...

2. The proposed converter

After all this overlong introductory text, it’s time to have a glance at the proposed converter, written as a custom writer in Lua... But, before proceeding further, let’s also try to understand what may have failed in the two first cases. The main problem, in this author’s opinion, is that they both tried something *without having the necessary yet building blocks* in SILE, and thus, without a clear design and vision. Underline and strikethrough? Bullet lists and enumerations? Cross-references? Tables? Code highlighting? Custom styles? None of them were really available, and some of these features are still missing from SILE’s standard distribution, as of version v0.12.5.⁶

Lessons learnt⁷, I tried to proceed the other way round, first implementing the building blocks found necessary, until *at last* tackling with Markdown. And since we now have this Pandoc “custom Lua writer” route, I went for it (using, for what is worth, their “Classic” API).

Eventually, the SILE-native route might be preferable in the long term. However, I do believe that one has to seek the low-hanging fruits first. Some of the choices made here are therefore debatable⁸, but in the end, the *user* needs a solution that produces quality results. The internal implementation can change later and be improved, one way or another. In other terms, I thrive for functionality over an ideal implementation that might never be reachable. What *end-users* need is a working solution to process their Markdown content into a neat PDF. That’s one of the key point in *software adoption* —that is decently does what you would expect, without having to dabble into obscure details⁹.

⁶ This author has dabbled into bullet lists and enumerations, tables, styling, cross-references, strikethrough, etc. with some proposals made to the core distribution. I do not say this for “bragging”, but rather to reinforce the idea that when one gets all the building blocks available at hand, things then come in a more natural fashion and the pieces of the puzzle start to assemble.

⁷ The above comments might sound harsh to the individuals who made the previous attempts. Let’s not be unfair, though: their efforts paved the way to the new attempt discussed here. Also, my own approach can be wrong too. It’s an **opinionated** viewpoint, after all!

⁸ E.g. syntax highlighting is currently based on a somehow deprecated solution.

⁹ I cannot even imagine the efforts that would be required to fix the existing `lua-`

3. Supported elements

Obviously sectioning works.

More or sectioning

Unnumbered sections work too.

3.1. Basic typesetting

Paragraphs are of course supported. Lorem ipsum dolor, sit amet consetetur, sadipscing elitr, sed diam non umy eirmod tempor, invidunt ut labor.

As of formatting, *italic*, **bold**, and monospace all work as expected.

Unicode is supported. ☺

Pandoc comes natively with smart typography: 3 dashes in Markdown are converted to an em-dash (—), 2 dashes to an en-dash for ranges (ex., “it’s all in chapters 12–14”). Note that Pandoc also provides the necessary support for smart quotes and apostrophes. Three dots are converted to an ellipsis (...)

Just to test that text sequences that need to be escaped in SILE language are properly managed, here is some text with %, \ and { } characters.

A backslash or two spaces at the end of the line in the Markdown source force a line break.

Superscripted or subscripted text: H₂O is a liquid, 2¹⁰ is 1024.

Some ~~deleted text~~, some underlined text, and some SMALL CAPS.

3.2. Lists

3.2.1. Enumerations and bullet lists

Itemized list:

- Item 1
- Item 2

Numbered lists:

1. First item
2. Second item

List nesting:

mark-based attempt and ensure it is good at supporting all of the rich Pandoc-flavor of Markdown... As for installing a Pandoc fork, this is pushing too much strain on the user, in my opinion.

1. Item
 - Hello
 - a. Hey
 - b. Ho
 - World
2. Item...
 - ... With several paragraphs
 - Fruits
 - Apple
 - Orange
 - Vegetables
 - Carrot
 - Potato

And here we go back to that item.

3.2.2. Task lists

☐ Unchecked item

☒ Checked item

3.2.3. Numbered example lists

Also known as continuous lists.

- (1) First example
- (2) Second example

Some paragraph text...

- (3) Third example
- (4) Fourth example is good.

As the good example (4) illustrates, numbered list work well.

3.2.4. Definition lists

Let's be honest, the current implementation is sort of a hack for now.

apples

Good for making applesauce.

oranges

Citrus!

3.3. Block quotes

Block quotes are written like so.

They can span multiple paragraphs, if you like.

And can be nested.

More nesting...

This is a block quote.

1. This is a list inside a block quote.
2. Second item.

3.4. Line blocks

In “line blocks”, this converter interprets the indentation as quad kerns. For instance:

Line one.

Line too.

Line three.

Line four.

3.5. Rules

An horizontal rule:

Just to check how it spans over the line, here is an orizontal rule in a block quote:

Lorem ipsum dolor sit amet Lorem ipsum dolor sit amet Lorem ipsum
dolor sit amet Lorem ipsum dolor sit amet Lorem ipsum dolor sit amet

3.6. Links and footnotes

Here is a link to the SILE website.

Here is link to a section heading in the current document (3.1) and to another (3.9) with custom identifier

They might not be visible in the PDF output, but hover them and click. They work.

I am not sure yet what a link to a local document should do.

Here's a footnote¹⁰.

¹⁰ Some footnote text.

3.7. Languages

Beside the global language setting in the metadata preamble (3.15), language changes within the text are supported, either around blocks or inline elements. It is not much visible below, obviously, but the language setting affects the hyphenation and other properties. In the case of French, for instance, you can see the special thin space before the exclamation point.

Cette citation est en français !

Or inline in text: « Encore du français ! »

3.8. Images

Here is an image:



An image has a legend if it stands on its own paragraph.

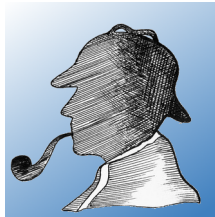


Figure 1. Sherlock Holmes

👉 SVG is supported too.

You can specify the required image width and height, as done just above actually, the SVG image being resized at `height=0.6em` — Note that it accepts any unit system supported by SILE.



Figure 2. A “manicule” ornament

3.9. Tables

Pandoc extends Markdown with several methods for declaring table. Obviously, it’s a question of input syntax, and the converter shouldn’t have to bother. Let us check anyway the generated tables look good in all cases.

3.10. Simple table

Right	Left	Center	Default
12	12	12	12
123	123	123	123
1	1	1	1

TABLE 1. Demonstration of a simple table.

Simple headerless table without caption:

12	12	12	12
123	123	123	123
1	1	1	1

3.10.1. Multiline table

Centered Header	Default	Right Aligned	Left Aligned
First	row	12.0	Example of a row that spans multiple lines.
Second	row	5.0	Here’s another one. Note it also spans multiple lines.

TABLE 2. Here is the caption.
It, too, may span multiple lines.

Multiline headerless table:

First	row	12.0	Example of a row that spans multiple lines.
-------	-----	------	---

Second	row	5.0	Here's another one that also spans multiple lines.
--------	-----	-----	--

TABLE 3. Here's a multiline table without a header.

3.10.2. Grid tables

Fruit	Price	Advantages
Bananas	\$1.34	<ul style="list-style-type: none"> • built-in wrapper • bright color
Oranges	\$2.10	<ul style="list-style-type: none"> • cures scurvy • tasty

TABLE 4. Sample grid table.

Grid table with alignments:

Right	Left	Centered
Bananas	\$1.34	built-in wrapper

Headerless grid table with alignments.

Right	Left	Centered
-------	------	----------

3.10.3. Pipe tables

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

TABLE 5. Demonstration of a pipe table.

3.11. Raw blocks

Raw blocks are Pandoc's way to pass native content to the underlying processor. This converter therefore supports a `{=sile}` annotation on code blocks, to pass through their content in SILE language, as shown below.

For instance, this sentence is typeset in a *raw block*, in SILE language.

Likewise, on an inline content: *idem*.

It also supports `{=sile-lua}` to pass Lua code, as in a SILE `\script`. This is just a convenience compared to the preceding one, but it allows you to exactly type its content as if it was in a code block (i.e. without having to bother wrapping it in a script).

This is called from Lua.

3.12. Code

Here's a Lua code sample:

```
SILE.registerCommand("section", function (options, content)
-- Registering a sectioning command is as simple as that with
-- our styles and sectioning packages.
options.style = "sectioning:section"
SILE.call("sectioning", options, content)
end, "Begin a new section.")
```

And a Python one:

```
import time
# Quick, count to ten!
for i in range(10):
    # (but not *too* quick)
    time.sleep(0.5)
    print(i)
```

3.13. Custom styles

The converter support the `{custom-style="..."}` syntax for custom styles. If such a named style exists, it is applied.

Here is some line block marked as “Poetry”:

*I met a lady in the meads,
Full beautiful—a faery's child,
Her hair was long, her foot was light,
And her eyes were wild.*

And some inline **message**, marked as “Warning”.

Obviously, these styles do not exist by default, so you have to provide a “style declaration” in some external script, and ensure it is loaded when SILE processes the document. Refer to the `scripts` entry in the metadata preamble (3.15).

3.14. Bibliography citations

Support is minimal in this version. At some later stage, we may want to consider using SILE’s bibliography package... For now, the converter just extracts the information and tries to present it decently.

Here is a “cite” (Doe 2011, 67), another (Doe 2011) and yet another (Doe 2011, 67; Smith 2012, 234–256).

3.15. Metadata preamble

Pandoc provides various ways to specify document metadata (e.g., with a YAML metadata block). This converter supports the following properties.

3.15.1. General properties

These document-wise properties are common with other converters:

- `papersize`: defaults to “a4”;
- `lang`: a BCP 47 language identifier such as “en-US”; only the country code is used by SILE and defaults to “en”;
- `dir`: writing direction (either “ltr” or “rtl”, ignored otherwise, and left absent if unspecified —implying “ltr”).

3.15.2. PDF metadata properties

These properties, also supported by several other converters, will here be used as PDF metadata.

- `title`: a string;
- `author`: a string or list;
- `subject`: a string;
- `keywords`: a string or list.

3.15.3. SILE-specific properties

SILE-specific formatting and layout settings are structured in a `sile` object property, which can contain:

- `class`: the SILE class to use (for now, defaulting to `omibook`)
- `font-family`: the main font family, as a string or a list —in the latter case, the additional fonts are declared as fallbacks for characters that would not be found in the main font;
- `font-size`: the main font size, as a number or string (which can refer to a SILE unit);

- `scripts`: a list of additional scripts that SILE will load after the main packages —so that you can override some definitions, provide custom styles, etc.

There are several reasons for introducing a `sile` namespace, rather than using the same options as other Pandoc writers (e.g. a `mainfont` entry, as used by HTML or LaTeX):

- To avoid cluttering inappropriate properties at a global level.
- To support more things than some writers (e.g. font fallbacks),
- To be able to select different settings for SILE and other writers (e.g. `documentclass` for LaTeX and `sile.class` have no reason to contain the same thing... That could occur for very basic class names which are likely to exist in both solutions —say, perhaps “book”— but that’s all);
- To avoid the mess with the loose *ad hoc* properties from some other writers (e.g. LaTeX’s `monofont`, `sansfont`, etc. are somewhat ill-designed). We use the class and style paradigms with more elegance.

4. Unsupported elements

4.1. Math equations

Inline math equation: NO SUPPORT YET ($\omega = d\phi / dt$). Display math should get its own line like so:

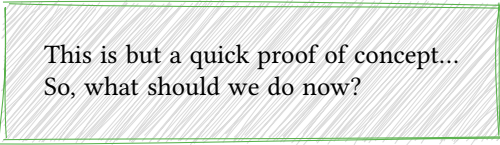
NO SUPPORT YET ($I = \int \rho R^2 dV$)

4.2. Extensions and other features...

Several topics are left to a later version of this converter:

- Pandoc templates, so we can make some Pandoc CLI options work, such as `-s` (standalone), `--toc`, etc.
- “Standalone mode” will actually be necessary to better control some features, e.g. metadata etc. should not be overridden if the output is intended to be used not as a standalone document but as an include file in some other document.
- Table of contents (and probably too, lists of figures/tables).
- Paragraph indents. The converter currently sets `\neverindent`. Paragraph skips too should probably be addressed. There are some (small but annoying) challenges there, so we went on without them —it’s all about reaching the low-hanging fruits again, see *supra*.

5. Towards a conclusion?



This is but a quick proof of concept...
So, what should we do now?

If you read so far, it's maybe up to you, my gentle reader, to provide feedback and help pushing this experiment farther.