cmarker.typ

https://github.com/SabrinaJewson/cmarker.typ

This package enables you to write CommonMark Markdown, and import it directly into Typst.

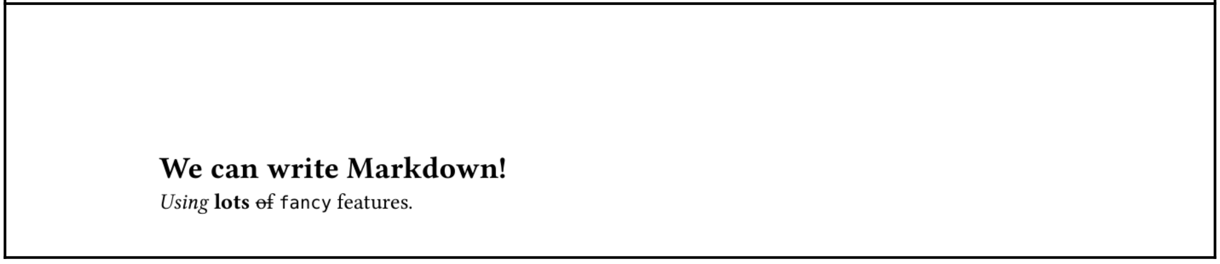
simple.typ

```
#import "@preview/cmarker:0.1.6"

#cmarker.render(read("simple.md"))
```

simple.md

```
# We can write Markdown!

*Using* __lots__ ~of~ `fancy` [features]
(https://example.org/).
```

simple.pdf



We can write Markdown!

Using **lots** ~~of~~ `fancy` features.

This document is available in Markdown and rendered PDF formats.

1. API

We offer a two functions:

1. render
2. render-with-metadata

1.1. render

```
render(
  markdown,
  smart-punctuation: true,
  math: none,
  h1-level: 1,
  raw-typst: true,
  html: (:),
  label-prefix: "",
  prefix-label-uses: true,
  heading-labels: "github",
  scope: (:),
  show-source: false,
  blockquote: none,
) -> content
```

The parameters are as follows:

- `markdown`: The CommonMark Markdown string to be processed. Parsed with the pulldown-cmark Rust library. You can set this to `read("somefile.md")` to import an external markdown file; see the documentation for the read function.
  - Accepted values: Strings and raw text code blocks.
  - Required.
- `smart-punctuation`: Automatically convert ASCII punctuation to Unicode equivalents:

- nondirectional quotations (" and ') become directional (“” and ‘’);
- three consecutive full stops (...) become ellipses (…);
- two and three consecutive hypen-minus signs (-- and ---) become en and em dashes (– and —).

Note that although Typst also offers this functionality, this conversion is done through the Markdown parser rather than Typst.

- Accepted values: Booleans.
- Default value: `true`.

- `math`: A callback to be used when equations are encountered in the Markdown, or `none` if it should be treated as normal text. Because Typst does not support LaTeX equations natively, the user must configure this.
  - Accepted values: Functions that take a boolean argument named `block` and a positional string argument (often, the `mitex` function from the mitex package), or `none`.
  - Default value: `none`.

  For example, to render math equation as a Typst math block, one can use:

  ```
  #import "@preview/mitex:0.2.4": mitex
  #cmarker.render(`$\int_1^2 x \mathrm{d} x$`, math: mitex)
  ```

  which renders as: $\int_1^2 x \, \mathrm{d}x$
- `h1-level`: The level that top-level headings in Markdown should get in Typst. When set to zero, top-level headings are treated as titles, ## headings become = headings, ### headings become == headings, et cetera; when set to 2, # headings become == headings, ## headings become === headings, et cetera.
  - Accepted values: Integers in the range [-128, 127].
  - Default value: 1.
- `raw-typst`: Whether to allow raw Typst code to be injected into the document via HTML comments. If disabled, the comments will act as regular HTML comments.
  - Accepted values: Booleans.
  - Default value: `true`.

  For example, when this is enabled, `<!--raw-typst #circle(radius: 10pt) -->` will result in a circle in the document (but only when rendered through Typst). See also `<!--typst-begin-exclude-->` and `<!--typst-end-exclude-->`, which is the inverse of this.
- `html`: The dictionary of HTML elements that cmarker will support.
  - Accepted values: Dictionaries whose keys are the tag name (without the surrounding <>) and whose values can be:
    - `("normal", (attrs, body) => [/* … */])`: Defines a normal element, where `attrs` is a dictionary of strings, `body` is content, and the function returns content.
    - `("void", (attrs) => [/* … */])`: Defines a void element (e.g. `<br>`, `<img>`, `<hr>`).
    - `("raw-text", (attrs, body) => [/* … */])`: Defines a raw text element (e.g. `<script>`, `<style>`), where `body` is a string.
    - `("escapable-raw-text", (attrs, body) => [/* … */])`: Defines an escapable raw text element (e.g. `<textarea>`), where `body` is a string.
    - `(attrs, body) => [/* … */]`: Shorthand for `("normal", (attrs, body) => [/* … */])`.
  - Default value: `(:)`.
  - Overridable keys: The following HTML elements are provided by default, but you are free to override them: `<sub>`, `<sup>`, `<mark>`, `<h1>`–`<h6>`, `<ul>`, `<ol>`, `<li>`, `<dl>`, `<dt>`, `<dd>`, `<table>`,

  `<thead>`, `<tfoot>`, `<tr>`, `<th>`, `<td>`, `<hr>`, `<a>`, `<em>`, `<strong>`, `<s>`, `<code>`, `<br>`, `<blockquote>`, `<figure>`, `<figcaption>`, `<img>`.

  For example, the following code would allow you to write `<circle radius="25">` to render a 25pt circle:

  ```
  #cmarker.render(read("input.md"), html: (
    circle: ("void", attrs => circle(radius: int(attrs.radius) * 1pt))
  ))
  ```

- `label-prefix`: If present, any labels autogenerated by footnotes and headings will be prefixed by this string. This is useful to avoid collisions.
  - Accepted values: A valid label or the empty string.
  - Default value: The empty string.
- `prefix-label-uses`: Whether references to labels such as `[@label]` and `[link](#label)` should be prefixed by `label-prefix`.
  - Accepted values: Booleans.
  - Default value: `true`.
- `heading-labels`: How the cases of labels autogenerated by headings should be transformed. The label prefix (given in `label-prefix`) will **not** be transformed.
  - Accepted values:
    - `"github"`: Match GitHub's slugification algorithm: strip out invalid characters, lowercase everything, convert ASCII spaces to hyphens, and number duplicate headings to avoid collisions. For example, `# My heading!` will become `<my-heading>`.
    - `"jupyter"`: Match Jupyter's slugification algorithm: just convert ASCII spaces to hyphens. For example, `# My heading!` will become `<My-heading!>`. Unlike the real Jupyter, we also number duplicate headings; but this shouldn't have any difference in practice since Jupyter doesn't allow referring to duplicate headings anyway.
  - Default value: `"github"`.
- `scope`: A dictionary providing the context in which the evaluated Typst code runs. It is useful to pass values in to code inside `<!--raw-typst-->` blocks, but can also be used to override element functions generated by cmarker itself.
  - Accepted values: Any dictionary.
  - Default value: `(:)`.
  - Overridable keys:
    - All built-in Typst functions.
    - `rule`: Expected to be a function returning content. Will be used when thematic breaks (`---` in Markdown) are encountered. Defaults to `line.with(length: 100%)`.
- `show-source`: A debugging tool. When set to `true`, the Typst code that would otherwise have been displayed will be instead rendered in a code block.
  - Accepted values: Booleans.
  - Default value: `false`.
- `blockquote`: Deprecated! This used to control how blockquotes were rendered, but we now default to `quote(block: true)`. If you want to override how blockquotes look, either use `#show quote.where(block: true)` or use `scope: (quote: …)`.
  - Accepted values: Functions accepting content and returning content, or `none`.
  - Default value: `none`.

This function returns the rendered `content`.

1.2. render-with-metadata

```
render-with-metadata(
  markdown,
  smart-punctuation: true,
  math: none,
  h1-level: 1,
  raw-typst: true,
  html: (:),
  label-prefix: "",
  prefix-label-uses: true,
  heading-labels: "github",
  scope: (:),
  show-source: false,
  blockquote: none,
  metadata-block: none,
) -> (metadata, content)
```

The `render-with-metadata` functions works the same as `render` with two exceptions:

1. This function returns `(meta, body)`. This allows the user to freely manipulate the metadata.

   ```
   #let (meta, body) = render-with-metadata(input)
   // `body` will be the same as:
   #let body = render(input)
   ```
2. This function has an extra argument: `metadata-block`. The other arguments are the same as `render`.

- `metadata-block`: How to parse the metadata block. Only a single metadata block at the beginning of the document is supported; metadata blocks not at the start will be ignored. If `none`, the metadata block will not be parsed. The behaviour is the same as using `render`.
  - Accepted values:
    - `"frontmatter-raw"`: Parse the metadata block and return it as a string.
    - `"frontmatter-yaml"`: Parsed the metadata block as YAML and return it as a dictionary.
    - `none`.
  - Default value: `none`.

2. Resolving Paths Correctly

Because of how Typst handles paths, elements like images will by default resolve relative to the project root of `cmarker` itself and not your project.

To fix this, one can override the `image` function in the scope the Typst code is evaluated.

```
#import "@preview/cmarker:0.1.6"

#cmarker.render(
  read("yourfile.md"),
  scope: (image: (path, alt: none) => image(path, alt: alt))
)
```

3. References, Labels, Figures and Citations

`cmarker.typ` integrates well with Typst's native references and labels. Where in Typst you would write `@foo`, in Markdown you write `[@foo]`; where in Typst you would write `@foo[Chapter]`, in Markdown you'd write `[Chapter][@foo]`. You can also write `[some text](#foo)` to have "some text" be a link that points at `foo`. Headings are automatically given references according to their name, but lowercased and with spaces replaced by hyphens:

```markdown
# My nice heading

We can make a link to [this section](#my-nice-heading).
We can refer you to [@my-nice-heading]. <!-- generates “We can refer you to Section 1.” -->
Or to [Chapter][@my-nice-heading]. <!-- generates “We can refer you to Chapter 1.” -->
```

If you have two headings with the same title, they'll be numbered sequentially in the fashion of GitHub Markdown: `heading`, `heading-1`, `heading-2`, etc.

If you want to cite something from a bibliography, you can do it much the same way: `[@citation]`.

If you want to customize the label of a heading from the default, you can use `id=` in HTML, or use `raw-typst` blocks (note that these **must** be preceded by a newline):

```html
<h1 id="my-nice-section">My nice heading</h1>
```

```markdown
# My other nice heading
<!--raw-typst <my-other-nice-section> -->

See [@my-nice-section] and [@my-other-nice-section].
<!-- generates “See Section 1 and Section 2.” -->
```

This can also be used with figures, which is very powerful:

```markdown
Please refer to [@my-graph]. <!-- generates “Please refer to Figure 1.” -->

<figure id="my-graph">

![a graph](graph.png)</figure>
```

If you encounter collision errors across multiple Markdown files, you can set the `label-prefix` option. For example, setting `label-prefix: "file-a-"` will convert the label of `# My Heading` into `file-a-my-heading`. By default, references within the file will expect the unprefixed name; this can be changed by setting `prefix-label-uses: false`.

4. Supported Markdown Syntax

We support CommonMark with a couple extensions.

- Paragraph breaks: Two newlines, i.e. one blank line.
- Hard line breaks (used more in poetry than prose): Put two spaces at the end of the line.
- `*emphasis*` or `_emphasis_`: *emphasis*
- `**strong**` or `__strong__`: **strong**
- `~strikethrough~`: ~~strikethrough~~
- `[links](https://example.org)`: links
- `### Headings`, where `#` is a top-level heading, `##` a subheading, `###` a sub-subheading, etc
- `` `inline code blocks` ``: `inline code blocks`

-
  ~~~
  ```
  out of line code blocks
  ```
  ~~~

  Syntax highlighting can be achieved by specifying a language after the opening backticks:

  ~~~
  ```rust
  let x = 5;
  ```
  ~~~

giving:

```
let x = 5;
```

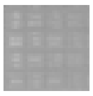
- `---`, making a horizontal rule. This corresponds to the Typst code `#rule()`, which, if not overridden by the `scope` parameter, defaults to `#line(length: 100%)`:

-
  ```
  - Unordered
  - lists
  ```

  ‣ Unordered
  ‣ Lists

-
  ```
  1. Ordered
  1. Lists
  ```

  1. Ordered
  2. Lists
- `$x + y$` or `$$x + y$$`: math equations, if the `math` parameter is set.
- `> blockquotes`, if the `blockquote` parameter is set.
- Images: `![Some tiled hexagons](examples/hexagons.png)`, giving

  
- Tables:

  ```
  | Column 1 | Column 2 |
  | -------- | -------- |
  | Row 1 Cell 1 | Row 1 Cell 2 |
  | Row 2 Cell 1 | Row 2 Cell 2 |
  ```

Column 1	Column 2
Row 1 Cell 1	Row 1 Cell 2
Row 2 Cell 1	Row 2 Cell 2

- Footnotes:

  ```
  Some text[^footnote]
  [^footnote]: content
  ```
- HTML, e.g. `<sub>subscript</sub>` for $_{\text{subscript}}$.

5. Interleaving Markdown and Typst

Sometimes, you might want to render a certain section of the document only when viewed as Markdown, or only when viewed through Typst. To achieve the former, you can simply wrap the section in `<!--typst-begin-exclude-->` and `<!--typst-end-exclude-->`:

```
<!--typst-begin-exclude-->
Hello from not Typst!
<!--typst-end-exclude-->
```

Most Markdown parsers support HTML comments, so from their perspective this is no different to just writing out the Markdown directly; but `cmarker.typ` knows to search for those comments and avoid rendering the content in between.

Note that when the opening comment is followed by the end of an element, `cmarker.typ` will close the block for you. For example:

```
> <!--typst-begin-exclude-->
> One

Two
```

In this code, "Two" will be given no matter where the document is rendered. This is done to prevent us from generating invalid Typst code.

Conversely, one can put Typst code inside a HTML comment of the form `<!--raw-typst […]-->` to have it evaluated directly as Typst code (but only if the `raw-typst` option to `render` is set to `true`, otherwise it will just be seen as a regular comment and removed):

```
<!--raw-typst Hello from #text(fill:blue)[Typst]!-->
```

6. Limitations

Although I tried my best to escape everything correctly, I won't provide a hard guarantee that everything is fully sandboxed even if you set `raw-typst: false`. That said, Typst itself is well-sandboxed anyway.

7. FAQ

7.1. Typst is saying it can't find my image – it's looking inside `cmarker` for some reason!

See Resolving Paths Correctly.

7.2. How do I include multiple Markdown files in one project?

See the multi-file example.

Note that this example uses a *single* call to `render` with concatenated Markdown, i.e. `cmarker.render(read("file-a.md") + read("file-b.md"))`. If you instead wish to use `cmarker.render(read("file-a.md"))` followed by `cmarker.render(read("file-b.md"))`, you may encounter collision issues if two headings have the same name. This can be resolved in two ways:

- Giving the headings an explicit ID: `<h1 id="my-id">My Heading</h1>` instead of `# My Heading`.
- Setting a label prefix, e.g. `label-prefix: "my-file-"`.

7.3. My image file contains spaces, but it gets rendered as text!

This is a Markdown quirk – `![alt](image path.png)` is seen as plain text instead of an image. To fix it, use `![alt](<image path.png>)`.

7.4. My Markdown after an open HTML tag is getting rendered as text!

Another Markdown quirk – in code like `<p>hello _world_</p>` or `<!-- -->hello _world_`, italics will not be generated.

There are two fixes: either insert some empty inline-level HTML at the start, e.g. `<span></span><p>hello _world_</p>`, or insert two newlines after the opening tag:

```
<p>

hello _world_</p>
```

7.5. How do I centre tables? How do I have them span the page width?

This can be achieved via show rules. For example, centring:

```
#show table: t => align(center, t)
#cmarker.render("
  | a | b |
  | - | - |
  | c | d |
")
```

And having them span the page width:

```
#show table: t => {
  if t.columns.all(c => c == auto) {
    table(columns: (1fr,) * t.columns.len(), align: t.align, ..t.children)
  } else {
    t
  }
}
```

8. Development

- Build the plugin with `./build.sh`, which produces the `plugin.wasm` necessary to use this.
- Compile examples with `typst compile examples/{name}.typ`.
- Compile this README to PDF with `typst compile README.typ`.
- Run tests with `cargo test --workspace`. Consult `test-runner/main.rs` for a guide on writing tests.
- Fuzz the library with `cargo +nightly fuzz run fuzz`.