TiefDownConverter Documentation

Tiefseetauchner et al.

January 05, 2026

Contents

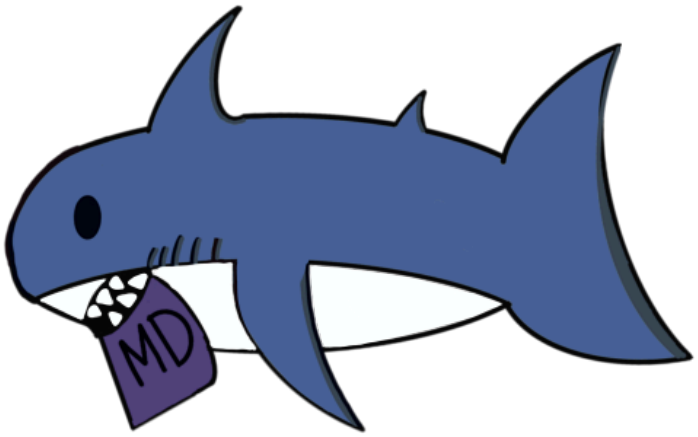

Figure 1: TiefDownConverter Mascot

1. LICENSE

```
MIT License

Copyright (c) 2025 Lena Tauchner

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

2. The what?

If you want to skip the funny written explanations, skip to the > Usage section.

Well, that's a good question. Ask it later.

Jk, of course you may ask it now. TiefDown is a project format I made up to make it easier to convert my files (mostly Markdown, but any type works) into something pretty. As a matter of fact, this documentation is managed by a TiefDown project!

The important thing is that this isn't a markdown parser, replacement or anything like that. It's a project format, and it's not even a format, it's pretty much just a manifest file and an executable.

2.1. Why?

I wonder myself every day. But alas, I should know, I wrote this cluster**** so let me explain. The initial concept was born from pain (as many are). I was pretty tired of exporting my files, then converting them, overwriting my old files, then converting them again, overwriting all history in the process. It was just... a mess.

So I did what any sane person would do: I learned Python.

Well, I'm being facetious. I didn't "learn Python," I just expanded my capabilities to calling programs from the command line.

So my script, at first, just called Pandoc, then pdflatex, and then pdflatex again for good measure. It created a PDF, overwriting my old one. It was basically just converting a single markdown file into a PDF with a basic TeX template (in my case, LiX Novel).

Then I realized that writing a 40-chapter story in a single markdown file was even dumber than whatever I made in Python. So I added a little combination logic. In the process, I had to write Lua filters as well, and then I added versioning, and then I added conversion to multiple different PDFs, and then I added EPUB support and-you know what? That was a dumb idea. The Python script soon reached 200 lines of code, which was untenable.

So yeah, I decided to make a new book. And of course - ***everything*** broke. Instantly. I had to copy and paste things, adjust my Python script, rewrote it a bit, and boom - suddenly I had two different projects with different processes, different outputs, different versions, different everything.

And then... I started a third book. Aaaand the Python script didn't really fulfill my needs, so I rewrote it in Bash. But worse.

I thought I had it all figured out. With Python. Then Bash. Then I started a short story and lost my ******* mind.

2.2. How, oh wise programmer, did you solve this problem?

I'm glad you asked! I'm glad. I... I hope you asked? Well, regardless of whether or not you did, I'll tell you.

I learned Rust.

For real this time, I learned a completely new programming language just for this. But there was a reason, or a few rather:

1. I wanted cross-platform support.
2. I wanted a single executable.
3. I needed a language with good CLI support because, believe it or not, I'm *awful* at GUIs.
4. I'm crazy.

These reasons led me to two options: Python, a language I was somewhat familiar with but didn't particularly enjoy writing in, and Rust, a language I had never written in before but was very interested in.

Evidently, I chose Rust.

So I started: a CLI interface, command-line calls, and so on. Here's the rundown of how it works internally:

- You initialize a project with `tiefdownconverter init`. This creates a few bits and bobs, but importantly the `manifest.toml` file. This contains all the information needed to actually manage and convert the project.
- You can then manipulate the project, and so on.
- When you add your files to the input directory, running `tiefdownconverter convert` will do a few things. In order to make it easy to understand, here's an explanation I would understand (had I not written it):
  - Take input files
  - Convert them
  - Convert them again (No one knows why)
  - ???
  - Profit

Isn't that simple?

It isn't. But oh well. We've got a lot of work to do on this, and if you're interested, don't shy away from the Contributing section!

2.3. So, what's the point?

Really? Making my life easier. I wanted to export my novel as a PDF in A4, 8x5in, so on. If I can make your life easier as well, then I'm the happiest woman alive.

2.4. Use Cases

So where does TiefDownConverter actually come in handy? Well, anywhere you need Markdown to turn into something nice without manually fiddling with formats every time. Here are a few scenarios where it saves the day:

- **Writing books** - Really handy if you are already writing in the likes of Markdown, and you really quickly need an a4 copy for printing a proof copy, a 8x5 in copy because you print with a random provider and a weird other format because you're from the US and A4 paper hasn't been invented there yet.
- **Technical Documentation** - I love looking at pretty documentation, and writing it in Markdown simplifies my workflow greatly. And once I'm done writing my documentation, I want it both as an HTML and as a PDF, so people can print it and put it in the "never read" shelf.
  Oh and I guess I could also print a book from it but eh eh eh, don't give me ideas.
- **Blogging sites** - No.
- **Blogging sites pretty please?** - Okay, maybe you *could* use TDC to make a dynamic project that automatically creates a blog for you. Maybe you *could* use CustomPreprocessors conversion to create named files. Maybe you *could* set up tooling around this. But you *shouldn't*.
  Trust me. I tried. (For fun to be clear)

Basically, if your workflow involves Markdown and you're sick of manually converting everything, TiefDown is your new best friend.

If you had a best friend in the first place, which, I don't.

2.5. Support

Now, you want support? Check out the Discord or write an issue on GitHub!

- Discord Server° (https://discord.gg/EG3zU9cTFx)
- GitHub Issues° (https://github.com/Tiefseetauchner/TiefDownConverter/issues)

3. Usage

The basic usage of `tiefdownconverter` is relatively simple. The difficult part is understanding the templating system and how to customise it for your use cases. Presets can only do so much.

Note: I wrote this paragraph before the big refactor. The basic usage is no longer simple.

3.1. Installation

Currently the only way to install `tiefdownconverter` is to either build it yourself, install it from cargo, or download a precompiled binary from the releases page°. Then just add it to the path and you're good to go. You can of course also just call it relatively by placing the binary in your project folder or something like that.

If you build from source, run `cargo build [--release]` or `cargo install --path .`.

That said, the recommended way to install TiefDownConverter is `cargo install tiefdownconverter`. This will always install the latest version.

Downloading from the release is as simple as downloading the appropriate version (Windows, Mac, Linux) and adding it to a folder in the path. You could also add tiefdownconverter to a folder and run it from there.

There are a few dependencies that you need to install.

- Pandoc°: Conversion from Markdown to TeX, Typst and Epub.
- A TeX distribution: For converting TeX files to PDF. It has to include xelatex.
  - If using TeX Live° you may need to additionally install `texlive-xetex` depending on your system.
  - If using MikTeX°, no need to do anything.
- Typst°: For converting Typst files to PDF.

Windows is easy enough: `winget install miktex pandoc typst`.

Linux varies by distro of course, but for ubuntu it's `apt install texlive-xetex pandoc` and `cargo install typst` or downloading the typst binary and adding it to the path.

Mac is still to be tested, but MacTex should have XeTeX installed.

Now you should be able to run `tiefdownconverter` from the command line. You can test it by initialising a test project using `tiefdownconverter init testproject` and running `tiefdownconverter convert` in the project directory or `tiefdownconverter convert -p testproject`. You could also, as a test, clone the Github Repo° and run `tiefdownconverter convert -p docs`.

3.2. Getting started

TL;DR: Make a folder, go into it and run `tiefdownconverter init` and `tiefdownconverter convert`. That's it.

Long answer: First off, you need to create a project using `tiefdownconverter init`. This will create a new project **in the current directory**. You can (and maybe should) specify a project, like `tiefdownconverter init your_project`.

This command creates the basic template structure like so:

```
your_project/
├── Markdown/
│   └── Chapter 1 - Introduction.md
├── template/
│   ├── meta.tex
```

```
6 │   └── template.tex
7 └── manifest.toml
```

The Markdown folder contains an example Markdown file. When placing your markdown files in this folder, make sure they're named like `... XX ... .md`, with anything following the number being ignored. *This is important*, as the converter will use this to sort the files for conversion, as otherwise it'd have no idea in which order they should be converted. Essentially, the first number you include must be the number of the file in order, so I suggest using a pattern like `Chapter X.md` or `X - ... .md`.

Now you should be able to run `tiefdownconverter convert -p path/to/your_project` (or omitting the -p flag if you're already in the project directory) and it should generate a PDF file in the project directory using the default LaTeX template. You can now adjust the template, add your own input files (Markdown or otherwise), and so on.

3.3. The input "directory"

Your source files are the main input for the converter, and as such their structure is important. The converter will look for files in the `Markdown` directory (or the directory specified during project creation) and will sort them by a chapter number. Namely, your files should be named `Whatever X Whatever else.ext`, where X is a number (you don't have to name them 01, 02 etc., as we parse the number as an integer, leading zeros are removed). The converter will then sort them by the first number and combine them in that order regardless of extension.

You can also add subdirectories in the input directory. These will be combined after the file with the same number. For example, consider the following directory structure:

```
1 Markdown/
2 ├── Chapter 1 - Introduction.md
3 ├── Chapter 2 - Usage.md
4 ├── Chapter 2 - Usage/
5 │   ├── Chapter 1 - Usage detail 1.md
6 │   └── Chapter 2 - Usage detail 2.md
7 └── Chapter 3 - Customisation.md
```

The converter will combine the files in the following order:

1. Chapter 1 - Introduction.md
2. Chapter 2 - Usage.md
3. Chapter 2 - Usage/Chapter 1 - Usage detail 1.md
4. Chapter 2 - Usage/Chapter 2 - Usage detail 2.md
5. Chapter 3 - Customisation.md

That is, the converter orders a directory by the same logic as other files (and even does so recursively), and directories are combined after the file with the same number.

You can change what directory the converter looks for markdown files in by changing the `markdown_dir` field in the manifest.toml file or saying `-m path/to/markdown/dir` when initialising the project.

3.4. Markdown projects

Now, above is a simplified explanation. If you want the full picture, read on.

With version 0.8.0 and above, the converter can handle multiple markdown folders at the same time. This is called a "markdown project" and it is the most convoluted way to think about markdown directories. Basically, a TiefDown project can have multiple markdown projects, that

are loaded as described above. But they have additional information stored in them, importantly **markdown project specific metadata**.

Now, why does this exist? Well, the basic idea is that you can have multiple projects per project. Markdown projects per TiefDown project, that is. It's useful for books for example, where you may have shared templates and metadata (like an author) but separate content and metadata (like a title) for the different books. This, in theory, simplifies the workflow substantially - but makes it more complicated to understand.

First off, the setup. You can run

```bash
tiefdownconverter project markdown add <PROJECT_NAME> <PATH_TO_MARKDOWN_DIR>
<PATH_TO_OUTPUT_DIR>
```

to add a markdown project to a TiefDown project. Per default, this is either not set at all, using the default markdown directory and output directory, or it is set to the default markdown directory and output directory of the TiefDown project, which are `Markdown` and `.` respectively. Importantly, the output directory is relevant for the conversion - it is used to separate the templating for the different projects, as well as the markdown files. So don't use the same output directory for multiple projects unless you hacked TDC to change the output format to include the template name, in which case, tell me how you did it.

The output directory is also important as the templates are all saved to the same file name per default (as in, the template output file name), and if you didn't use a different output directory, you'd overwrite the generated output for the other project. (Unless, as I said, you found a workaround that doesn't involve a PR.)

Project specific metadata is interesting as well, as in the end, it is merged with the shared metadata. So when you run the conversion, first the shared metadata is loaded and then the markdown project specific metadata, overwriting the shared metadata.

Setting metadata is done by using the meta command, similarly to the shared-meta command, except that you have to specify the markdown project name as well. As an example, you may run

```bash
tiefdownconverter project markdown meta <PROJECT_NAME> set <KEY> <VALUE>
```

to set a metadata value for a markdown project.

You can also assign resources, which are files that are copied to the compile directory from the markdown project directory *and are ignored during the conversion process*. This is done by using the resource command. For example, if you had multiple books as separate markdown projects, you could have a `cover.png` file for each book separately and then use the resource management to copy it to be able to be used in a template, for example an epub or as the cover of a PDF. Check out the resources command for more information.

You can also assign a profile to a markdown project which, if I may say so myself as the person who needed it, is awesome.

Imagine... Well, don't imagine. Look at this documentation on github. You can see, there's a markdown project called `cli_markdown` and a markdown project called `man_markdown`. They both contain relatively similar but different markdown files but importantly, they act quite different. One generates the manpage, and one the documentation you are reading right now. These are two completely different tasks, so the `man_markdown` project uses a different profile per default.

A default profile is assigned using the `--default-profile` flag. This is the profile that will be used to convert the markdown project *by default*. That doesn't mean you can't use all templates as you

wish, you can always use the `--profile` flag to specify a different profile or the `--templates` flag to specify a different set of templates.

The convert command also accepts a list of markdown projects using the -m flag. The provided templates or profile are then only converted for the specified markdown projects.

3.5. Input Processing

Input processing converts source files into a format your template includes. This happens via preprocessors. Input files are grouped by extension, and each group is processed in one shot by the matching preprocessor (default or custom). The converter concatenates the stdout of these runs and writes it to the configured combined output file.

By default, LaTeX templates use `output.tex` and Typst templates use `output.typ`. When you assign preprocessors to a template, you also specify the combined output filename.

Preprocessors can be extension-specific. See Preprocessing for details.

3.6. Customising the template

The key idea behind tiefdownconverter is that it can handle multiple templates at the same time. This is done by creating a template file in the template directory and adding it to the project's manifest.toml file.

You could do this manually, if you were so inclined, but using `tiefdownconverter project template <TEMPLATE_NAME> add` is much easier. Check the Usage Details and specifically the templates add command for the usage of this command. But importantly, once you created the template and added it to the manifest, you will be able to convert using it. `tiefdownconverter convert -p path/to/your_project --templates <TEMPLATE_NAME>` will convert only the selected template, aiding in debugging.

And now, you're pretty much free to do whatever you want with the template. Write tex or typst templates, use custom filters, so on.

3.7. Adjusting template behaviour

You have a few options for editing template behaviour using `tiefdownconverter`. You can of course edit the template files directly, but there are a few more options.

Mainly and most interestingly, lua filters can adjust the behaviour of the markdown conversion. These are lua scripts that are run by pandoc before the markdown is converted to tex or typst. You can add lua filters to a template by either editing the manifest or using `tiefdownconverter project templates <TEMPLATE_NAME> update --add-filters <FILTER_NAME>`. This can be either the path to a lua filter (relative to the project directory) or a directory containing lua filters. Look up Lua Filters for more information

You can also change the name of the exported file by setting the `output` option. For example, `tiefdownconverter project templates <TEMPLATE_NAME> update --output <NEW_NAME>`. This will export the template to `<NEW_NAME>` instead of the default `<TEMPLATE_NAME>.pdf`. This field is required where the output extension isn't knowable, so for Custom Preprocessors/Processor conversions.

Similarly, you could change the template file and type, though I advice against it, as this may break the template. I advice to just add a new template and remove the old one using `tiefdownconverter project templates <TEMPLATE_NAME> remove`.

3.8. Conversion Profiles

A conversion profile is a shortcut to defining templates for the conversion. If you're dealing with a lot of templates, you may be considering only converting some at any time - for example, web ready PDFs vs. print ready PDFs, or only converting a certain size of PDF.

For that, there are conversion profiles which simply are a list of templates. It's essentially like saving your –templates arguments.

You can create these profiles with the `project profile add` command, setting a name and a comma seperated list of templates. Removing a profile is also possible with the `project profile remove` command.

Running a conversion with a profile is as simple as adding the `--profile` flag.

The manifest file contains a section for this, if you desire to configure them manually:

```toml
[[profiles]]
name = "PDF"
templates = ["PDF Documentation LaTeX", "PDF Documentation"]
```

Conversion profiles can also be set as a default for a Markdown Project, which will by default only convert the templates in the profile when converting that project. That means, when running TDC without a `--templates` or `--profile` argument, it will use the templates in the assigned profile only.

3.9. Writing templates

Importantly, when you write your own template, you need to include the content somehow. That somehow is done via `\input{output.tex}` or `#include "./output.typ"`. This will include the output of the Markdown conversion in your template file. If you're using custom preprocessors, you can change the output file of the conversion. See Preprocessing for more information. For CustomPreprocessors conversion, this is the output file already, as there is no template file. Should you be using CustomProcessor conversion, the combined file is AST and not really usable, so don't think about it. See custom processor conversion for more information.

3.10. Shared Metadata

Metadata is a key part of any project. That's why TiefDown allows adding project wide metadata as well as per markdown project metadata (see Markdown Projects). This makes sharing metadata not only between templates easier, but even between projects.

Imagine you want to manage four books. Each of them has a different author, but the same publisher. You could add the publisher to the metadata of each project, but that would be a lot of work, especially if the publisher changes branding. Instead, you can add the publisher to the shared metadata, and then add the author to the metadata of each project.

To add metadata to a project, use the `tiefdownconverter project shared-meta set` command. This writes the metadata to the project's manifest.toml file and when converting the project, the metadata will be written to respective metadata files for the template type (e.g. metadata.tex for LaTeX and metadata.typ for Typst) or be used to replace arguments during the conversion. You can then import these files in your template and access the metadata.

3.10.1. Accessing metadata in LaTeX

Metadata, per default, is accessed in LaTeX via the `\meta` command. This command takes a key and returns the value of that key. However, accessing undefined values is undefined behaviour. Be careful to only access metadata that is defined, and double check, I never know what happens when

you access undefined metadata. It may just throw an error, it may also write random characters to your document.

3.10.2. Accessing metadata in Typst

Much nicer than LaTeX, Typst has a type system! Just import `meta` and access the keys on it. Though the linter will error out if you do this, so you can write your own `metadata.typ` in the template directory with placeholder values.

3.10.3. Using metadata for custom preprocessors and processors

Now, I mentioned argument replacement. Custom preprocessors or processors may include arguments like `{{title}}`, which, during conversion, are replaced with the metadata field `title` if available. That means that you can, for example, use `--title {{title}}` as an argument to a custom processor for a CustomProcessor template that converts to HTML to set the title field of said HTML file. It's more complicated, but if you know Pandoc, you know what I mean (I hope).

3.11. Epub Support

EPUB support in TiefDownConverter isn't as fancy as LaTeX or Typst, but you can still tweak it to look nice. You don't get full-blown templates, but you can mess with CSS, fonts, and Lua filters to make it work how you want. *This template type is however somewhat depricated. It will not be removed but there likely won't be any new features added to it.*

3.11.1. Customizing CSS

EPUBs use stylesheets to control how everything looks. Any `.css` file you drop into `template/my_epub_template/` gets automatically loaded.

For example, you can change the font, line height, and margins like so:

```css
body {
  font-family: "Noto Serif", serif;
  line-height: 1.6;
  margin: 1em;
}
blockquote {
  font-style: italic;
  border-left: 3px solid #ccc;
  padding-left: 10px;
}
```

3.11.2. Adding Fonts

Fonts go into `template/my_epub_template/fonts/`, and TiefDownConverter will automatically pick them up. To use them, you just need to reference them properly in your CSS:

```css
@font-face {
  font-family: "EB Garamond";
  font-style: normal;
  font-weight: normal;
  src: url("../fonts/EBGaramond-Regular.ttf");
}

body {
```

```
 9   font-family: "EB Garamond", serif;
10 }
```

This is a good time to mention, epub is just a zip file. As such, as it is generated by pandoc, it has a predefined structure, and you have to bend to that. Fonts are in a font directory, and stylesheets in a styles directory. Thus you have to *break out* of the styles directory with .. to get to the fonts directory. Keep that in mind, it took me a while to figure out.

3.11.3. Metadata and Structure

EPUBs need some basic metadata, which you define in the YAML front matter of your Markdown files. Stuff like title, author, and language goes here:

yaml

```yaml
---
title:
- type: main
text: "My Publication"
- type: subtitle
text: "A tale of loss and partying hard"
creator:
- role: author
text: Your Name
rights: "Copyright © 2012 Your Name"
---
```

You can also do this via the custom processor arguments, adding metadata as described in the pandoc documentation. For example, to use a separate metadata file, you can do this:

bash

```bash
tiefdownconverter project [PROJECT_NAME] processors add "Metadata for EPUB" -- --metadata-file metadata.yaml
tiefdownconverter project [PROJECT_NAME] templates <TEMPLATE_NAME> update --processor "Metadata for EPUB"
```

This will include the metadata file in the conversion process, removing the need for the YAML front matter in your Markdown files and allowing you to use different metadata files for different templates.

3.11.4. Using Lua Filters

Want to tweak the structure? That's what Lua filters are for. You can use them to rename chapters, remove junk, or modify how elements are processed.

Example: Automatically renaming chapter headers:

lua

```lua
function Header(el)
  if el.level == 1 then
    return pandoc.Header(el.level, "Chapter: " .. pandoc.utils.stringify(el.content))
  end
end
```

And that's it. You get a customized EPUB without having to fight with the defaults. Enjoy!

3.12. Conversion Engines

There are currently five ways to convert your files. All of them are based on the same system. The main difference is the output format and the program it gets converted with.

3.12.1. LaTeX

LaTeX is the best supported by TiefDownConverter, with the most presets. But as TiefDownConverter is a general-purpose Markdown to PDF converter, the format doesn't matter. LaTeX provides the highest degree of customization, making it ideal for structured documents, novels, and academic papers.

The primary way to interact with LaTeX is through templates. Lua filters and such are secondary, but an important part of the conversion process to adjust behavior for different document classes.

3.12.2. Typst

Typst is another supported engine, offering a more modern alternative to LaTeX with a simpler syntax and automatic layout adjustments. TiefDownConverter allows you to specify Typst templates in the project manifest.

Typst templates work similarly to LaTeX templates but are easier to modify if you need structured documents without deep LaTeX knowledge.

As far as I could tell, typst templates are also far more adherent to the general typst syntax, so Lua filters are not as important. But they can still be used to adjust the output, especially for more advanced use cases.

3.12.3. EPUB

TiefDownConverter also supports EPUB conversion, making it suitable for e-book generation. The conversion process uses Pandoc to transform the Markdown content into EPUB, applying any Lua filters defined in the manifest.

This however does not really support much in the way of templating. Customization should be done primarily via Lua filters.

However, you can still get some customization by including CSS and font files in your template folder. That's the reason epub has to have a folder in the first place, so you can place CSS and font files in there. Of course you can add multiple epub templates, but I don't know why you would want to.

EPUB output is particularly useful for digital publishing, ensuring compatibility with e-readers and mobile devices.

3.12.4. Custom Preprocessors Converter

Okay. Stick with me here. The idea is, you are already converting my input files with Pandoc, why not let me convert them to whatever format? Well, this is where Custom Preprocessors Conversion comes in. This long fabled feature is the most complicated one, and you need a deep understanding of how TiefDownConverter works and at least the ability to read Pandoc's documentation to even use it. But if you're willing to put in the effort, you can do some pretty cool things.

The basic idea is, just, let the user decide what pandoc does. The result is chaos.

I'm being facetious, but this is actually the most powerful way to customize the output. You add one or multiple preprocessors as described in Preprocessing and set the output path of the preprocessor and template to the same path. Then you can do whatever pandoc allows. Want to convert to RTF? No issue. But beware: you need to actually understand what's going on, otherwise you'll end up in implementation hell.

One important thing to keep in mind is to never, ever, try to generate a non-concatenateable format (like docx, pdf, ...) with CustomPreprocessors conversion. It won't work as soon as you have more than one input format. Use custom processor conversion instead.

3.12.5. Custom Processor Converter

If that wasn't bad enough: We've got more. Custom Processor Conversions are a way to combine multiple input files to a file type that isn't just a collection of lines. For example, take a docx file. It isn't just multiple simpler files strung together, it is a complicated web of zip files and openxml and all that jazz.

Now, why may that be an issue? Simply put: multiple input formats are converted in batches and strung together when using a custom preprocessor converter. Instead, we need to convert all output formats to a common type and merge them afterwards for the conversion to a docx to work.

That's what custom processor conversion is for. It uses preprocessors to convert all input files to a common format (Pandoc native AST) and combines these files to arrive at a single AST file, letting pandoc then convert to the required format using a custom processor.

Custom processors require an output file compatible with pandoc. When creating such a template, make sure to reference the pandoc guide. You can use custom preprocessors as usual, but you will need to set the output format flag (`-t`) to `native`. A custom processor can also be used when converting from AST to the output format. Any pandoc parameter is accepted, but the `-o` and `-f` flags are set at compile time and mustn't be added.

3.13. Writing filters

> **Note:** This section only really addresses LaTeX, but the concepts are the same for Typst and epub.

If you are in the business of writing filters (and don't just solve everything in TeX itself), I advice checking out the documentation at https://pandoc.org/lua-filters.html°. But here's a quick rundown of what you can do. For example, if you wanted to change the font of all block quotes, there's a few things you'd need to do. First off, in your template, you will need to define a font. It could look something like this:

tex
```tex
\usepackage{fontspec}
\newfontfamily\blockquotefont{Noto Sans}
```

Then, add a filter to your template as described above. The filter could look something like this:

lua
```lua
function BlockQuote(el)
  local tt_start = pandoc.RawBlock('latex', '\\blockquotefont\\small')

  table.insert(el.content, 1, tt_start)

  return el
end
```

Of course, you could just redefine the font in TeX but I think this is a bit more flexible. One use case that is quite important is to change the way chapters are handled for LiX. In case of LiX, they expect `\h{chapter_name}` instead of `\section`, which is the standard behaviour of pandoc. So when you create a LiX backed template, you have to add a filter to change that behaviour. Something like this:

lua
```lua
function Header(elem)
  if elem.level == 1 then
    return pandoc.RawBlock("latex", "\\h{" .. pandoc.utils.stringify(elem.content) .. "}")
```

```
  end
  if elem.level == 2 then
    return pandoc.RawBlock("latex", "\\hh{" .. pandoc.utils.stringify(elem.content) .. "}")
  end
  -- add more levels here if needed
end
```

3.14. Preprocessing

A "Preprocessor" is a stupid word for defining your own pandoc conversion parameters. You can (and should) use this to adjust the behaviour of the converter. For example, you could define a preprocessor to add `--listings` to the pandoc command. This is useful if you want to have reasonable code output in your pdf.

If no preprocessor is defined, the converter will use default pandoc parameters, converting to the intermediate output file (in case of LaTeX, this is `output.tex`). But if you, and this is a purely hypothetical scenareo, want to run a conversion on mixed input md and typ files, you can define a typst specific preprocessor that simply uses cat.

If you want to define a preprocessor, you can do so by running

bash
```
tiefdownconverter project templates <TEMPLATE_NAME> update \
  --preprocessors <PREPROCESSOR_NAMES,...> \
  --preprocessor-output <PREPROCESSOR_OUTPUT>
```

to assign it to a template and

bash
```
tiefdownconverter project pre-processors add <PREPROCESSOR_NAME> -- [CLI_ARGS]
```

to create a new preprocessor.

For example, if you want to add `--listings` to the pandoc command, you could do so by adding `--listings` to the preprocessor. But importantly, **this overwrites the default preprocessor**. Defaults from that (as few as they may be) won't get carried over to the conversion.

bash
```
tiefdownconverter project pre-processors add "Enable Listings" -- --listings
```

The manifest would look something like this:

toml
```
...

[[custom_processors.preprocessors]]
name = "Enable Listings"
cli_args = ["--listings"]

[[templates]]
filters = ["luafilters/chapter_filter.lua"]
name = "PDF Documentation LaTeX"
output = "docs_tex.pdf"
template_file = "docs.tex"
template_type = "Tex"

[templates.preprocessors]
preprocessors = ["Enable Listings"]
combined_output = "output.tex"

```

```
18 ...
```

Now, you may be able to spot a neato featureo: preprocessors are assigned in an array. That means, you can have multiple preprocessors per template. With this power however comes the responsibility to define extension filters on your preprocessors. This is an extension-only glob pattern set via the `--filter` option when creating the preprocessor.

bash

```
1 tiefdownconverter project pre-processors add "No typst conversion" --filter "typ" --cli "cat"
```

If no filter is provided, the preprocessor applies to all files. In your template, you can then define the preprocessor list as well as the combined output of the preprocessor. This is important, as this output is then passed to the conversion engine (or copied for Custom Preprocessors Conversion).

3.15. Custom Preprocessors Conversion

I already hinted at it in Custom Preprocessors Converter, but I'll go into more detail here. The idea is to run a preprocessor and just skip any further processing. Straight from pandoc to the output.

You can do this by first defining a preprocessor, for example:

bash

```
1 tiefdownconverter project pre-processors add "RTF Preprocessor" -- -t rtf
```

As you can see, we're outputting as an RTF file. This means we need to add a template that deals with the same output:

bash

```
1 tiefdownconverter project template "RTF Template" add -o documentation.rtf -t custompandoc
2 tiefdownconverter project template "RTF Template" update --preprocessors "RTF Preprocessor" --
  preprocessor-output "documentation.rtf"
```

And that's it. TiefDownConverter will run the preprocessor, which outputs to documentation.rtf, and then the templating system will copy that output to your directory. Hopefully. Did I mention that this is experimental? Yeah, so if you have issues, please report them. Even if you're thinking "this is not a bug, it's a feature". It likely isn't.

3.16. Custom Processor Arguments

You can define custom arguments for your processors. These are passed to the processor, so xelatex, typst, so on, on compilation. For example, if you needed to add a font directory to your typst conversion, you could do so by adding the following to your manifest:

toml

```
1  ...
2
3  [[custom_processors.processors]]
4  name = "Typst Font Directory"
5  processor_args = ["--font-path", "fonts/"]
6
7  [[templates]]
8  name = "PDF Documentation"
9  output = "docs.pdf"
10 processor = "Typst Font Directory"
11 template_file = "docs.typ"
12 template_type = "Typst"
13
14 ...
```

Or... Just use the command to create it.

```bash
tiefdownconverter project processor "Typst Font Directory" add -- --font-path fonts/
```

Then append it to a template.

```bash
tiefdownconverter project template "PDF Documentation" update --processor "Typst Font Directory"
```

This is especially useful with custom processor converters.

3.17. Injections

Injections are a simple way to manipulate a preprocessing step on a per template basis without having to mess with preprocessors. An injection contains files, which are inserted before the preprocessing step. As such, the rules of the preprocessor apply, including file name scoped rules.

Injections are defined once in a project and then reused per template. The template also decides where in the document the injection is placed. There are three ways a file can be injected into the document:

- Header injections get applied at the top of the document and are rendered as the first files in a conversion.
- Body injections are inserted in the document in accordance with the default sorting rule, and are thus mixed in.
- Footer injections are treated equivalently to header injections, but are instead placed at the end of the conversion list.

For example, setting a different chapter 3 per template is made possible by making a file like `3 - injection_pdf.md` and `3 - injection_epub.md` and add the injections to the corresponding documents.

A header could be used for a separate index conversion, allowing you to define a different header for different output formats. Footers function similarly.

An injection can be added using the `project injections` command:

```bash
tiefdownconverter project injections create "HTML Footer" html_footer.html
```

You can then add an injection to an existing template using the `--header-injections`, `--body-injections` and `--footer-injections` flags respectively.

```bash
tiefdownconverter project template "HTML Documentation" update --footer-injections "HTML Footer"
```

Keep in mind each category can contain multiple injections.

3.18. Smart Cleaning

Smart cleaning is a feature that is relatively simple. If you enable it in your manifest, it will automatically remove stale or old conversion directories.

Enable it with the `--smart-clean` and set the threshold with `--smart-clean-threshold`. The threshold is 5 by default.

You can also manually trigger a smart clean with `tiefdownconverter project smart-clean` or a normal clean with `tiefdownconverter project clean`. The latter will remove all conversion directories, while the former will only remove the ones that are older than the threshold.

4. Usage Details

Below are the usage details for the various commands. **Note:** These are autogenerated! For clearer documentation, please see the Usage section.

4.1. `tiefdownconverter`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.1.1. Usage:

```
TiefDownConverter manages TiefDown projects.
TiefDown is a project structure meant to simplify the conversion process from Markdown to PDFs.
TiefDownConverter consolidates multiple conversion processes and templating systems to generate a configurable set or subset of output documents.
It is not in itself a converter, but a wrapper around pandoc, xelatex and typst. As such, it requires these dependencies to be installed.

Usage: tiefdownconverter [OPTIONS] <COMMAND>

Commands:
  convert            Convert a TiefDown project. By default, it will convert the current directory.
  init               Initialize a new TiefDown project.
  project            Update the TiefDown project.
  check-dependencies  Validate dependencies are installed.
  help               Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')

  -V, --version
          Print version
```

4.1.2. Subcommands:

- convert
- init
- project
- check-dependencies

4.2. `tiefdownconverter convert`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.2.1. Usage:

```
Convert a TiefDown project. By default, it will convert the current directory.

Usage: tiefdownconverter convert [OPTIONS]

Options:
```

```
  -p, --project <PROJECT>
          The project to convert. If not provided, the current directory will be used.
  -v, --verbose
          Enable verbose output.
  -t, --templates <TEMPLATES>...
          The templates to use. If not provided, the default templates from the manifest file will be used. Cannot be used with --profile.
  -P, --profile <PROFILE>
          The conversion profile to use. Cannot be used with --templates.
  -m, --markdown-projects <MARKDOWN_PROJECTS>...
          The markdown projects to convert. If not provided, all markdown projects will be converted.
  -h, --help
          Print help
```

4.3. `tiefdownconverter init`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.3.1. Usage:

```
Initialize a new TiefDown project.

Usage: tiefdownconverter init [OPTIONS] [PROJECT]

Arguments:
  [PROJECT]
          The project to initialize. If not provided, the current directory will be used.

Options:
  -t, --templates <TEMPLATES>...
          The preset templates to use. If not provided, the default template.tex will be used.
          For custom templates, use the update command after initializing the project.
          If using a LiX template, make sure to install the corresponding .sty and .cls files from https://github.com/NicklasVraa/LiX. Adjust the metadata in template/meta.tex accordingly.


          [possible values: template.tex, booklet.tex, lix_novel_a4.tex, lix_novel_book.tex, template_typ.typ, default_epub]

  -v, --verbose
          Enable verbose output.

  -n, --no-templates
          Do not include the default templates. You will need to add templates manually with Update

  -f, --force
          Delete the project if it already exists.

  -m, --markdown-dir <MARKDOWN_DIR>
          The directory where the Markdown files are located. If not provided, Markdown/ will be used.

      --smart-clean
          Enables smart clean for the project with a default threshold of 5.
```

```
32          If the number of conversion folders in the project is above this threshold, old
   folders will be cleaned, leaving only the threshold amount of folders.
33
34      --smart-clean-threshold <SMART_CLEAN_THRESHOLD>
35          The threshold for smart clean. If not provided, the default threshold of 5 will
   be used.
36          If the number of conversion folders in the project is above this threshold, old
   folders will be cleaned, leaving only the threshold amount of folders.
37
38  -h, --help
39          Print help (see a summary with '-h')
```

4.4. `tiefdownconverter project`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.4.1. Usage:

```
 1 Update the TiefDown project.
 2
 3 Usage: tiefdownconverter project [OPTIONS] [PROJECT] <COMMAND>
 4
 5 Commands:
 6   templates        Add or modify templates in the project.
 7   update-settings  Update the project manifest settings.
 8   pre-processors   Manage the preprocessors of the project.
 9   processors       Manage the processors of the project.
10   profiles         Manage the conversion profiles of the project.
11   shared-meta      Manage the shared metadata of the project.
12   markdown         Manage the markdown projects of the project.
13   injections       Manage the injections of the project.
14   list-templates   List the templates in the project.
15   clean            Clean temporary files from the TiefDown project.
16   smart-clean      Clean temporary files from the TiefDown project, leaving only the threshold
   amount of folders.
17   help             Print this message or the help of the given subcommand(s)
18
19 Arguments:
20   [PROJECT]  The project to edit. If not provided, the current directory will be used.
21
22 Options:
23   -v, --verbose  Enable verbose output.
24   -h, --help     Print help
```

4.4.2. Subcommands:

- templates
- update-settings
- pre-processors
- processors
- profiles
- shared-meta
- markdown
- injections

- list-templates
- clean
- smart-clean

4.5. `tiefdownconverter project templates`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.5.1. Usage:

```
Add or modify templates in the project.

Usage: tiefdownconverter project templates [OPTIONS] <TEMPLATE> <COMMAND>

Commands:
  add     Add a new template to the project.
  remove  Remove a template from the project.
  update  Update a template in the project.
  help    Print this message or the help of the given subcommand(s)

Arguments:
  <TEMPLATE>  The template name to edit or add.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.5.2. Subcommands:

- add
- remove
- update

4.6. `tiefdownconverter project templates add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.6.1. Usage:

```
Add a new template to the project.
If using a preset template name, the preset will be copied to the template folder.
If using a custom template, make sure to add the respective files to the template folder.
Available preset templates are: template.tex, booklet.tex, lix_novel_a4.tex,
lix_novel_book.tex, template_typ.typ, default_epub

Usage: tiefdownconverter project templates <TEMPLATE> add [OPTIONS]

Options:
  -f, --template-file <TEMPLATE_FILE>
          The file to use as the template. If not provided, the template name will be used.

  -v, --verbose
          Enable verbose output.

  -t, --template-type <TEMPLATE_TYPE>
```

```
          The type of the template. If not provided, the type will be inferred from the template file.

          [possible values: tex, typst, epub, custom-preprocessors, custom-processor]

  -o, --output <OUTPUT>
          The output file. If not provided, the template name will be used.

      --filters <FILTERS>...
          The luafilters to use for pandoc conversion of this templates markdown.
          Luafilters are lua scripts applied during the pandoc conversion.
          You can add a folder or a filename. If adding a folder, it will be traversed recursively, and any .lua file will be added.
          See the pandoc documentation and 'Writing filters' of the TiefDownConverter documentation for more details.

      --preprocessors <PREPROCESSORS>
          The preprocessors to use for this template.
          A preprocessor defines the arguments passed to the pandoc conversion from the specified input format.
          Each input format can have at most one preprocessor. Multiple preprocessors for the same input format will lead to an error.
          There can be a preprocessor without an input format, which will be used if no other preprocessor matches the input format. Only one such preprocessor is allowed.
          If using a CustomPreprocessors template, at least one preprocessor is required.
          Preprocessors replace all arguments. Thus, with preprocessors, you need to define the output file and format.
          For templates, that is the file imported by the template.

      --preprocessor-output <PREPROCESSOR_OUTPUT>
          The output file of the preprocessor. If not provided, the template name with the appropriate ending will be used.
          This is the file the input gets converted to. When preprocessing the input files, the files will get converted, combined and written to this filename.

      --processor <PROCESSOR>
          The processor to use for this template.
          A processor defines additional arguments passed to the conversion command.
          For LaTeX and typst templates, this allows extending the respective conversion parameters.
          Processors are incompatible with CustomPreprocessors conversions. Use preprocessors instead.

      --header-injections <HEADER_INJECTIONS>...
          The injection to use for prepending to the preprocessing step.
          A header injection can define one or more files that will be prepended to the preprocessing step.
          Files in header injections get prepended in the order that they are defined in in the manifest.
          Duplicate files will be added twice.
          Injections have to be defined in the manifest.

      --body-injections <BODY_INJECTIONS>...
          The injection to use for inserting into the preprocessing step.
          A body injection can define one or more files that will be inserted into the preprocessing step.
```

```
          Files in body injections get inserted in accordance with the sorting algorithm.
          Duplicate files will be added twice.
          Injections have to be defined in the manifest.

      --footer-injections <FOOTER_INJECTIONS>...
          The injection to use for appending to the preprocessing step.
          A footer injection can define one or more files that will be appended to the preprocessing step.
          Files in header injections get appended in the order that they are defined in in the manifest.
          Duplicate files will be added twice.
          Injections have to be defined in the manifest.

      --multi-file-output
          Enables multi-file output conversion for the template.
          When enabling multi-file output, every input file will be converted to a corresponding output file.

      --output-extension <OUTPUT_EXTENSION>
          The extension used for multi-file output conversion.
          This is required for multi-file outputs.

      --meta-gen-feature <META_GEN_FEATURE>
          Defines the feature level of and whether metadata files should be generated.
          None disables metadata generation.
          NavOnly only enables navigation metadata generation and injection.
          MetadataOnly only enables manifest metadata generation and injection.
          Full enables full navigation and manifest metadata generation and injection.

          [possible values: none, full, nav-only, metadata-only]

      --nav-meta-gen-output <NAV_META_GEN_OUTPUT>
          The path to generate the navigation metadata to.
          Gets saved in the temporary compilation directory.

      --metadata-meta-gen-output <METADATA_META_GEN_OUTPUT>
          The path to generate the manifest metadata to.
          Gets saved in the temporary compilation directory.

      --meta-gen-format <META_GEN_FORMAT>
          The format to generate metadata in.
          Can be Json or None.
          YML metadata is always generated, JSON metadata is only needed if used by an external program.

          [possible values: none, json]

  -h, --help
          Print help (see a summary with '-h')
```

4.7. `tiefdownconverter project templates remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.7.1. Usage:

```
Remove a template from the project.

Usage: tiefdownconverter project templates <TEMPLATE> remove [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.8. `tiefdownconverter project templates update`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.8.1. Usage:

```
Update a template in the project.

Usage: tiefdownconverter project templates <TEMPLATE> update [OPTIONS]

Options:
      --template-file <TEMPLATE_FILE>
          The file to use as the template. If not provided, the template name will be used.

  -v, --verbose
          Enable verbose output.

      --template-type <TEMPLATE_TYPE>
          The type of the template. If not provided, the type will be inferred from the template file.
          Changing this is not recommended, as it is highly unlikely the type and only the type has changed. It is recommended to create a new template instead.

          [possible values: tex, typst, epub, custom-preprocessors, custom-processor]

      --output <OUTPUT>
          The output file. If not provided, the template name will be used.

      --filters <FILTERS>...
          The luafilters to use for pandoc conversion of this templates markdown.
          This replaces all existing filters.

      --add-filters <ADD_FILTERS>...
          The luafilters to use for pandoc conversion of this templates markdown.
          This adds to the existing filters.

      --remove-filters <REMOVE_FILTERS>...
          The luafilters to use for pandoc conversion of this templates markdown.
          This removes the filter from the existing filters.

      --preprocessors <PREPROCESSORS>
          The preprocessors to use for this template.
          A preprocessor defines the arguments passed to the pandoc conversion from the specified input format.
          Each input format can have at most one preprocessor. Multiple preprocessors for the same input format will lead to an error.
```

```
37          There can be a preprocessor without an input format, which will be used if no other
    preprocessor matches the input format. Only one such preprocessor is allowed.
38          If using a CustomPreprocessor template, at least one preprocessor is required.
39          Preprocessors replace all arguments. Thus, with preprocessors, you need to define
    the output file and format.
40          For templates, that is the file imported by the template.
41
42      --add-preprocessors <ADD_PREPROCESSORS>...
43          The preprocessors to use for this template.
44          This adds to the existing preprocessors.
45
46      --remove-preprocessors <REMOVE_PREPROCESSORS>...
47          The preprocessors to use for this template.
48          This removes the preprocessor from the existing preprocessors.
49
50      --preprocessor-output <PREPROCESSOR_OUTPUT>
51          The output file of the preprocessor. If not provided, the template name with the
    appropriate ending will be used.
52          This is the file the input gets converted to. When preprocessing the input files,
    the files will get converted, combined and written to this filename.
53
54      --processor <PROCESSOR>
55          The processor to use for this template.
56          A processor defines additional arguments passed to the conversion command.
57          For LaTeX and typst templates, this allows extending the respective conversion
    parameters.
58          Processors are incompatible with CustomPreprocessor conversions. Use preprocessors
    instead.
59
60      --header-injections <HEADER_INJECTIONS>...
61          The injection to use for prepending to the preprocessing step.
62          A header injection can define one or more files that will be prepended to the
    preprocessing step.
63          Files in header injections get prepended in the order that they are defined in in
    the manifest.
64          Duplicate files will be added twice.
65          Injections have to be defined in the manifest.
66
67      --body-injections <BODY_INJECTIONS>...
68          The injection to use for inserting into the preprocessing step.
69          A body injection can define one or more files that will be inserted into the
    preprocessing step.
70          Files in body injections get inserted in accordance with the sorting algorithm.
71          Duplicate files will be added twice.
72          Injections have to be defined in the manifest.
73
74      --footer-injections <FOOTER_INJECTIONS>...
75          The injection to use for appending to the preprocessing step.
76          A footer injection can define one or more files that will be appended to the
    preprocessing step.
77          Files in header injections get appended in the order that they are defined in in
    the manifest.
78          Duplicate files will be added twice.
79          Injections have to be defined in the manifest.
80
81      --multi-file-output <MULTI_FILE_OUTPUT>
```

```
          Enables multi-file output conversion for the template.
          When enabling multi-file output, every input file will be converted to a corresponding output file.

          [possible values: true, false]

      --output-extension <OUTPUT_EXTENSION>
          The extension used for multi-file output conversion.
          This is required for multi-file outputs.

      --meta-gen-feature <META_GEN_FEATURE>
          Defines the feature level of and whether metadata files should be generated.
          None disables metadata generation.
          NavOnly only enables navigation metadata generation and injection.
          MetadataOnly only enables manifest metadata generation and injection.
          Full enables full navigation and manifest metadata generation and injection.

          [possible values: none, full, nav-only, metadata-only]

      --nav-meta-gen-output <NAV_META_GEN_OUTPUT>
          The path to generate the navigation metadata to.
          Gets saved in the temporary compilation directory.

      --metadata-meta-gen-output <METADATA_META_GEN_OUTPUT>
          The path to generate the manifest metadata to.
          Gets saved in the temporary compilation directory.

      --meta-gen-format <META_GEN_FORMAT>
          The format to generate metadata in.
          Can be Json or None.
          YML metadata is always generated, JSON metadata is only needed if used by an external program.

          [possible values: none, json]

  -h, --help
          Print help (see a summary with '-h')
```

4.9. `tiefdownconverter project update-settings`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.9.1. Usage:

```
Update the project manifest settings.

Usage: tiefdownconverter project update-settings [OPTIONS]

Options:
      --smart-clean <SMART_CLEAN>
          Enables smart clean for the project with a default threshold of 5.
          If the number of conversion folders in the project is above the smart_clean_threshold, old folders will be cleaned, leaving only the threshold amount of folders.

          [possible values: true, false]
```

```
11
12   -v, --verbose
13           Enable verbose output.
14
15       --smart-clean-threshold <SMART_CLEAN_THRESHOLD>
16           The threshold for smart clean. If not provided, the default threshold of 5 will
   be used.
17           If the number of conversion folders in the project is above this threshold, old
   folders will be cleaned, leaving only the threshold amount of folders.
18
19   -h, --help
20           Print help (see a summary with '-h')
```

4.10. `tiefdownconverter project pre-processors`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.10.1. Usage:

```
 1 Manage the preprocessors of the project.
 2 A preprocessor defines the arguments passed to the pandoc conversion from markdown.
 3 If using a CustomPreprocessor template, a preprocessor is required.
 4 Preprocessors replace all arguments. Thus, with preprocessors, you need to define the output
   file and format.
 5 For templates, that is the file imported by the template.
 6 Preprocessors are incompatible with epub conversion. Use processors instead.
 7
 8 Usage: tiefdownconverter project pre-processors [OPTIONS] <COMMAND>
 9
10 Commands:
11   add     Add a new preprocessor to the project.
12   remove  Remove a preprocessor from the project.
13   list    List the preprocessors in the project.
14   help    Print this message or the help of the given subcommand(s)
15
16 Options:
17   -v, --verbose
18           Enable verbose output.
19
20   -h, --help
21           Print help (see a summary with '-h')
```

4.10.2. Subcommands:

- add
- remove
- list

4.11. `tiefdownconverter project pre-processors add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.11.1. Usage:

```
1 Add a new preprocessor to the project.
```

```
 2
 3 Usage: tiefdownconverter project pre-processors add [OPTIONS] <NAME> [-- <CLI_ARGS>...]
 4
 5 Arguments:
 6   <NAME>
 7           The name of the preprocessor to create.
 8
 9   [CLI_ARGS]...
10           The arguments to pass to the preprocessor.
11
12 Options:
13       --filter <FILTER>
14           The file extension filter for the preprocessor.
15           This defines which input files the preprocessor is applied to. If not provided, the
   preprocessor will be applied to all input files.
16           Allows glob patterns. Excludes the leading dot. Only matches the file extension.
17
18   -v, --verbose
19           Enable verbose output.
20
21       --cli <CLI>
22           The program to use as the preprocessor.
23           Requires cli arguments
24           Should Pandoc not be the required preprocessor for your use case, you can change the
   called cli program.
25
26   -h, --help
27           Print help (see a summary with '-h')
```

4.12. `tiefdownconverter project pre-processors remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.12.1. Usage:

```
 1 Remove a preprocessor from the project.
 2
 3 Usage: tiefdownconverter project pre-processors remove [OPTIONS] <NAME>
 4
 5 Arguments:
 6   <NAME>  The name of the preprocessor to remove.
 7
 8 Options:
 9   -v, --verbose  Enable verbose output.
10   -h, --help     Print help
```

4.13. `tiefdownconverter project pre-processors list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.13.1. Usage:

```
 1 List the preprocessors in the project.
 2
 3 Usage: tiefdownconverter project pre-processors list [OPTIONS]
```

```
 4
 5 Options:
 6   -v, --verbose  Enable verbose output.
 7   -h, --help     Print help
```

4.14. `tiefdownconverter project processors`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.14.1. Usage:

```
 1 Manage the processors of the project.
 2 A processor defines additional arguments passed to the conversion command.
 3 For LaTeX and typst templates, this allows extending the respective conversion parameters.
 4 For CustomProcessor templates, this allows adding custom pandoc parameters.
 5 Processors are incompatible with CustomPreprocessors conversions. Use preprocessors instead.
 6
 7 Usage: tiefdownconverter project processors [OPTIONS] <COMMAND>
 8
 9 Commands:
10   add     Add a new processor to the project.
11   remove  Remove a processor from the project.
12   list    List the processors in the project.
13   help    Print this message or the help of the given subcommand(s)
14
15 Options:
16   -v, --verbose
17           Enable verbose output.
18
19   -h, --help
20           Print help (see a summary with '-h')
```

4.14.2. Subcommands:

- add
- remove
- list

4.15. `tiefdownconverter project processors add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.15.1. Usage:

```
 1 Add a new processor to the project.
 2
 3 Usage: tiefdownconverter project processors add [OPTIONS] <NAME> [-- <PROCESSOR_ARGS>...]
 4
 5 Arguments:
 6   <NAME>              The name of the processor to create.
 7   [PROCESSOR_ARGS]...  The arguments to pass to the processor.
 8
 9 Options:
10   -v, --verbose  Enable verbose output.
11   -h, --help     Print help
```

4.16. `tiefdownconverter project processors remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.16.1. Usage:

```
Remove a processor from the project.

Usage: tiefdownconverter project processors remove [OPTIONS] <NAME>

Arguments:
  <NAME>  The name of the processor to remove.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.17. `tiefdownconverter project processors list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.17.1. Usage:

```
List the processors in the project.

Usage: tiefdownconverter project processors list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.18. `tiefdownconverter project profiles`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.18.1. Usage:

```
Manage the conversion profiles of the project.
A conversion profile defines a collection of templates to be converted at the same time.
This can be used to prepare presets (for example, web export, PDF export, ...).
It can also be used for defining default templates for markdown projects.

Usage: tiefdownconverter project profiles [OPTIONS] <COMMAND>

Commands:
  add     Add a new conversion profile to the project.
  remove  Remove a conversion profile from the project.
  list    List the conversion profiles in the project.
  help    Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.18.2. Subcommands:

- add
- remove
- list

4.19. `tiefdownconverter project profiles add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.19.1. Usage:

```
Add a new conversion profile to the project.

Usage: tiefdownconverter project profiles add [OPTIONS] <NAME> [TEMPLATES]...

Arguments:
  <NAME>          The name of the profile to create.
  [TEMPLATES]...  The templates to add to the profile.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.20. `tiefdownconverter project profiles remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.20.1. Usage:

```
Remove a conversion profile from the project.

Usage: tiefdownconverter project profiles remove [OPTIONS] <NAME>

Arguments:
  <NAME>  The name of the profile to remove.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.21. `tiefdownconverter project profiles list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.21.1. Usage:

```
List the conversion profiles in the project.

Usage: tiefdownconverter project profiles list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.22. `tiefdownconverter project shared-meta`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.22.1. Usage:

```
Manage the shared metadata of the project.
This Metadata is shared between all markdown projects.
When converting, it is merged with the markdown project specific metadata.
When using the same key for shared and project metadata, the project metadata overrides the shared metadata.

Usage: tiefdownconverter project shared-meta [OPTIONS] <COMMAND>

Commands:
  set     Add or change the metadata. Overrides previous keys.
  remove  Remove metadata.
  list    List the metadata.
  help    Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.22.2. Subcommands:

- set
- remove
- list

4.23. `tiefdownconverter project shared-meta set`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.23.1. Usage:

```
Add or change the metadata. Overrides previous keys.

Usage: tiefdownconverter project shared-meta set [OPTIONS] <KEY> <VALUE>

Arguments:
  <KEY>    The key to set.
  <VALUE>  The value to set.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.24. `tiefdownconverter project shared-meta remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.24.1. Usage:

```
Remove metadata.

Usage: tiefdownconverter project shared-meta remove [OPTIONS] <KEY>

Arguments:
  <KEY>  The key to remove.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.25. `tiefdownconverter project shared-meta list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.25.1. Usage:

```
List the metadata.

Usage: tiefdownconverter project shared-meta list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.26. `tiefdownconverter project markdown`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.26.1. Usage:

```
Manage the markdown projects of the project.
A markdown project defines the markdown conversion process for a project.
There can be multiple markdown projects with different markdown files.
Each markdown project also has a seperate output folder ('.' per default).
A markdown project can have seperate metadata.
A markdown project can have resources that are copied to the respective conversion folder.

Usage: tiefdownconverter project markdown [OPTIONS] <COMMAND>

Commands:
  add        Add a new markdown project to the project.
  update     Update a markdown project in the project.
  meta       Manage the metadata of a markdown project.
  resources  Manage the resources of a markdown project.
  remove     Remove a markdown project from the project.
  list       List the markdown projects in the project.
  help       Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
```

```
24          Print help (see a summary with '-h')
```

4.26.2. Subcommands:

- add
- update
- meta
- resources
- remove
- list

4.27. `tiefdownconverter project markdown add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.27.1. Usage:

```
Add a new markdown project to the project.

Usage: tiefdownconverter project markdown add [OPTIONS] <NAME> <PATH> <OUTPUT>

Arguments:
  <NAME>    The name of the markdown project to create.
  <PATH>    The path to the markdown project.
  <OUTPUT>  The output folder.

Options:
      --default-profile <DEFAULT_PROFILE>  The default profile to use for converting this
  project.
  -v, --verbose                            Enable verbose output.
  -h, --help                               Print help
```

4.28. `tiefdownconverter project markdown update`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.28.1. Usage:

```
Update a markdown project in the project.

Usage: tiefdownconverter project markdown update [OPTIONS] <NAME>

Arguments:
  <NAME>  The name of the markdown project to update.

Options:
      --path <PATH>                        The path to the markdown project.
  -v, --verbose                            Enable verbose output.
      --output <OUTPUT>                    The output folder.
      --default-profile <DEFAULT_PROFILE>  The default profile to use for converting this
  project.
  -h, --help                               Print help
```

4.29. `tiefdownconverter project markdown meta`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.29.1. Usage:

```
Manage the metadata of a markdown project.
This metadata is markdown project specific and is not shared between projects.
This metadata takes precedence over the shared metadata.

Usage: tiefdownconverter project markdown meta [OPTIONS] <NAME> <COMMAND>

Commands:
  set     Add or change the metadata. Overrides previous keys.
  remove  Remove metadata.
  list    List the metadata.
  help    Print this message or the help of the given subcommand(s)

Arguments:
  <NAME>
          The name of the markdown project to update.

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.29.2. Subcommands:

- set
- remove
- list

4.30. `tiefdownconverter project markdown meta set`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.30.1. Usage:

```
Add or change the metadata. Overrides previous keys.

Usage: tiefdownconverter project markdown meta <NAME> set [OPTIONS] <KEY> <VALUE>

Arguments:
  <KEY>    The key to set.
  <VALUE>  The value to set.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.31. `tiefdownconverter project markdown meta remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.31.1. Usage:

```
Remove metadata.

Usage: tiefdownconverter project markdown meta <NAME> remove [OPTIONS] <KEY>

Arguments:
  <KEY>  The key to remove.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.32. `tiefdownconverter project markdown meta list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.32.1. Usage:

```
List the metadata.

Usage: tiefdownconverter project markdown meta <NAME> list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.33. `tiefdownconverter project markdown resources`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.33.1. Usage:

```
Manage the resources of a markdown project.
Resources are a way to include meta information and resources on a per project basis.
This is helpful for example for including a custom css file for a project, as that is not possible purely with metadata.
Resources are stored in the markdown folder and copied to the conversion directory for that profile before conversion.

Usage: tiefdownconverter project markdown resources [OPTIONS] <NAME> <COMMAND>

Commands:
  add     Add a new resource to the project.
  remove  Remove a resource from the project.
  list    List the resources in the project.
  help    Print this message or the help of the given subcommand(s)

Arguments:
  <NAME>
          The name of the markdown project to update.

Options:
  -v, --verbose
          Enable verbose output.

```

```
  -h, --help
          Print help (see a summary with '-h')
```

4.33.2. Subcommands:

- add
- remove
- list

4.34. `tiefdownconverter project markdown resources add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.34.1. Usage:

```
Add a new resource to the project.

Usage: tiefdownconverter project markdown resources <NAME> add [OPTIONS] [-- <PATHS>...]

Arguments:
  [PATHS]...  The paths to the resources. Seperated by spaces.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.35. `tiefdownconverter project markdown resources remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.35.1. Usage:

```
Remove a resource from the project.

Usage: tiefdownconverter project markdown resources <NAME> remove [OPTIONS] <PATH>

Arguments:
  <PATH>  The path to the resource.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.36. `tiefdownconverter project markdown resources list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.36.1. Usage:

```
List the resources in the project.

Usage: tiefdownconverter project markdown resources <NAME> list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
```

```
  7   -h, --help     Print help
```

4.37. `tiefdownconverter project markdown remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.37.1. Usage:

```
 1 Remove a markdown project from the project.
 2
 3 Usage: tiefdownconverter project markdown remove [OPTIONS] <NAME>
 4
 5 Arguments:
 6   <NAME>  The name of the markdown project to remove.
 7
 8 Options:
 9   -v, --verbose  Enable verbose output.
10   -h, --help     Print help
```

4.38. `tiefdownconverter project markdown list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.38.1. Usage:

```
 1 List the markdown projects in the project.
 2
 3 Usage: tiefdownconverter project markdown list [OPTIONS]
 4
 5 Options:
 6   -v, --verbose  Enable verbose output.
 7   -h, --help     Print help
```

4.39. `tiefdownconverter project injections`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.39.1. Usage:

```
 1 Manage the injections of the project.
 2 An injection defines an additional, template scoped mechanism for adding files to the combined
   output of the preprocessors.
 3 Each injection can have multiple files or directories associated with it.
 4 An injection can be used in three ways:
 5 - Header injections: Get preprended to the document in the order in which they are listed in
   the manifest.
 6 - Body injections: Get inserted and sorted in the primary document.
 7 - Footer injections: Get appended to the document in the order in which they are listed in
   the manifest.
 8
 9 Usage: tiefdownconverter project injections [OPTIONS] <COMMAND>
10
11 Commands:
12   add        Creates a new injection.
13   remove     Removes an injection.
```

```
  add-files  Adds files to an injection.
  list       List the injections in the project.
  help       Print this message or the help of the given subcommand(s)

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.39.2. Subcommands:

- add
- remove
- add-files
- list

4.40. `tiefdownconverter project injections add`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.40.1. Usage:

```
Creates a new injection.
Fails if an injection with that name already exists.

Usage: tiefdownconverter project injections add [OPTIONS] <NAME> [FILES]...

Arguments:
  <NAME>
          The name of the injection to create.
          Must be unique per project.

  [FILES]...
          The files to be used for the injections.
          Can be a directory.
          The order of the files here defines the order for header and footer injections.
          For body injections, the files are ordered as per the default algorithm.
          Files in directories are ordered as per the default algorithm.
          Duplicate files will be added twice.

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.41. `tiefdownconverter project injections remove`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.41.1. Usage:

```
Removes an injection.

Usage: tiefdownconverter project injections remove [OPTIONS] <NAME>

Arguments:
  <NAME>  The name of the injection to remove.

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.42. `tiefdownconverter project injections add-files`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.42.1. Usage:

```
Adds files to an injection.

Usage: tiefdownconverter project injections add-files [OPTIONS] <NAME> [FILES]...

Arguments:
  <NAME>
          The name of the injection to modify.

  [FILES]...
          The files to be added to the injection.
          Can be a directory.
          The order of the files here defines the order for header and footer injections.
          For body injections, the files are ordered as per the default algorithm.
          Files in directories are ordered as per the default algorithm.
          Duplicate files will be added twice.

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.43. `tiefdownconverter project injections list`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.43.1. Usage:

```
List the injections in the project.

Usage: tiefdownconverter project injections list [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.44. `tiefdownconverter project list-templates`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.44.1. Usage:

```
List the templates in the project.

Usage: tiefdownconverter project list-templates [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.45. `tiefdownconverter project clean`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.45.1. Usage:

```
Clean temporary files from the TiefDown project.

Usage: tiefdownconverter project clean [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

4.46. `tiefdownconverter project smart-clean`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.46.1. Usage:

```
Clean temporary files from the TiefDown project.
If the number of conversion folders in the project is above this threshold, old folders will be cleaned, leaving only the threshold amount of folders.
The threshold is set to 5 by default, and is overwritten by the threshold in the manifest.

Usage: tiefdownconverter project smart-clean [OPTIONS]

Options:
  -v, --verbose
          Enable verbose output.

  -h, --help
          Print help (see a summary with '-h')
```

4.47. `tiefdownconverter check-dependencies`

Version: `tiefdownconverter 0.11.0-ALPHA.1`

4.47.1. Usage:

```
Validate dependencies are installed.

```

```
Usage: tiefdownconverter check-dependencies [OPTIONS]

Options:
  -v, --verbose  Enable verbose output.
  -h, --help     Print help
```

5. Contributing

This project is open source, and I'd love for you to contribute! There's a few things you should know before you start.

> **NOTE:** When raising the version, don't forget to change the version in the documentation as well!!!

5.1. The architecture

The project has a relatively straight forward conversion process, as shown in the diagram below.

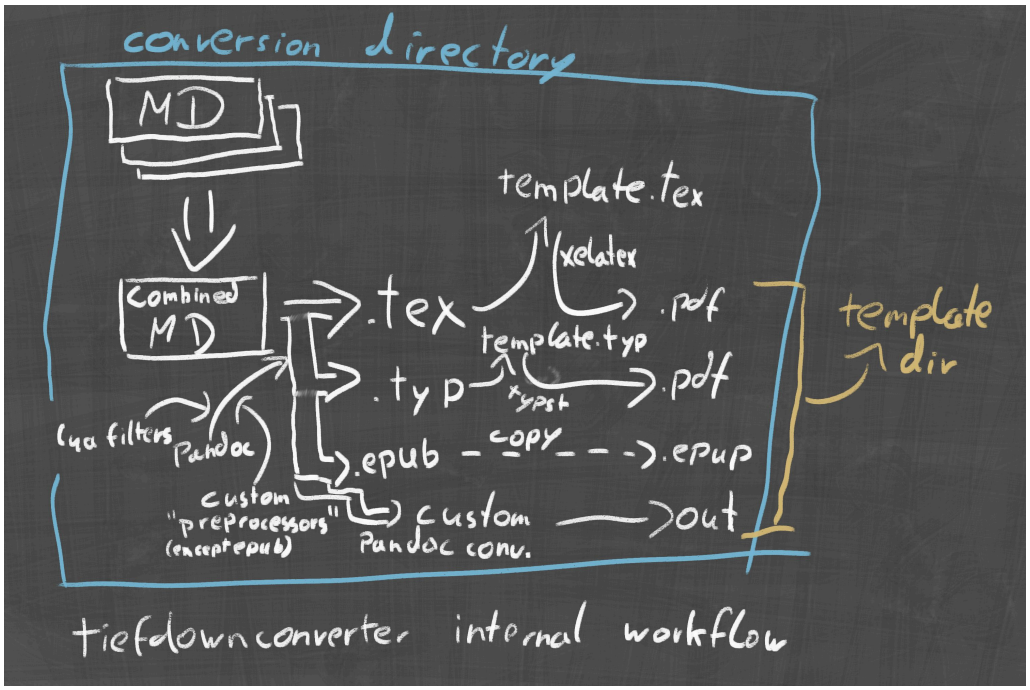

Figure 2: Conversion Process

5.2. Pull Requests

Pull Requests should be made with either a link to an issue or an explanation of

1. What was the problem
2. How is it solved now
3. How did it affect the documentation

It takes a lot of work to understand the intention of code you didn't write and then judging whether this was indeed the intended outcome. That's why it's helpful for everyone if there's an explanation on what was changed and why.

5.3. Conversion

Conversion is split in a few different steps:

1. Combine all the markdown files into one megafile called `combined.md`.
2. Run Pandoc conversion to TeX, EPUB, or Typst. This uses Lua filters that are defined in the `manifest.toml` file.
3. Run XeLaTeX on all TeX templates, Typst on all Typst templates, and so on.

Say you were to add a new conversion type. In `converters.rs`, you'd need to add a new function that handles the full conversion. Including handling lua filters, markdown conversion, so on. This con-

verter function has to then be included in our conversion decision logic in `conversion_decider.rs`. And for that you need to add a new TemplateType, which includes editing the implementations. Then you need to add the new template type decision logic to `get_template_type_from_path`.

5.4. Presets

> **NOTE:** This is a bit of a niche use case, so documentation is lacking. You can always ask for help on this in a GitHub issue.

You can also add new presets, but that's a bit more involved. You should check the implementation for the existing presets, I don't think it's useful to document this niche use case for now.

5.5. Manifest

Hope you don't have to change the manifest.toml file. If you do, change the manifest model, increase the version number in `consts.rs` and add a upgrade logic to `manifest_model.rs`.

5.6. Tests

Currently primarily integration tests. See the `tests` folder for examples. Any pull request to main will automatically run tests, and the expectation is that at least the existing tests work. If they break, fix your code or, if you changed behavior on purpose, the tests.

I appreciate it if you add test coverage for your changes. I especially would appreciate more unit tests, but the tests I have are sufficient for now. Integration tests take priority over unit tests for me, as the overall behavior is more important to me than the individual functions, and I only have so much time that I want to spend on this project.

5.7. Documentation

When changing the documentation, it is of utmost importance that the documentation outputs are correctly generated. *These are not automatically generated on release* but rather held in git to more easily track changes during a pull request.

To make sure this documentation is up to date, consider whether your changes significantly affect the workflow of using TiefDownConverter. If you add a command or flag, make sure to run `tools/generate_docs.py`. Either way, when changing the documentation, always run `tiefdownconverter convert -p docs` before committing the changes.

Praise be you don't need to have any fonts installed anymore - they are packaged in the fonts directory. But if you happen to change the fonts, you will need to replace them in fonts/ and add your own fonts to the templates.

5.8. Code Style

I don't have one. I'm sorry.