TiefDownConverter Documentation

Tiefseetauchner et al.

August 28, 2025

Contents

1 Shared Metadata

Shared metadata is defined once for the whole project and is available to every markdown project. It lives under `[shared\_metadata]` in the manifest file and is merged with project specific metadata at conversion time. Values defined in a markdown project override entries from the shared metadata.

Use shared metadata for information that stays the same across multiple books or documents, like the publisher or an overarching author list.

2 Introduction

For the documenation of the library, see docs.rs.

For the documentation of the CLI, see TiefDownConverter.

This is the documentation for the TiefDown concepts. This won't explain the library or the CLI usage, but rather function as an introduction to the basics of TiefDown for users and contributors alike.

2.1 What is TiefDown?

TiefDown is a project format for managing markdown files and converting them to other formats. It's not a markdown parser, but rather a project format and management system.

Importantly, the project is split in a few parts:

- The `manifest.toml` file, which contains all the information needed to manage and convert the project.
- The `template` directory, which contains all the templates for the project.
- One or more markdown directories, corresponding to markdown projects.

3 Metadata Settings

Metadata settings influence how metadata files are generated. The `[metadata\_settings]` table currently supports the `metadata\_prefix` option. This prefix determines the name of the macro or object used to access metadata in templates.

For example, with

```
[metadata_settings]
metadata_prefix = ”book”
```

the generated LaTeX file defines a `\\book\{\}` command while Typst exposes a `book` object. In other words, the prefix fully replaces the default `meta` name. If no prefix is set the command and object are called `meta`.

4 Manifest File

The manifest file is the heart of the project. It contains all the information needed to manage and convert the project.

It consists of a few important parts (for the full documentation, check `https://docs.rs/tiefdownlib/latest/tiefdownlib/manifest_model/index.html`):

- A version number
  - This is used to determine if the manifest is compatible with the current version of TiefDownConverter. If it's not, the manifest will be automatically updated in the process of loading. Newer versions of the manifest file are rejected by the implementation.
- The automatic smart clean flag
  - This is a boolean flag that determines if the project should be cleaned automatically when a conversion is run. This is useful for projects that are constantly being updated, allowing a user to decide how many conversion folders they want to keep.
- The smart clean threshold
  - This is the number of conversion folders that are kept before the oldest ones are deleted.
- A list of markdown projects
- A list of templates
- A custom processors object
- A table of shared metadata for all markdown projects
- A metadata settings object
- A list of profiles available for the conversion

5 Conversion folders

The conversion folder is the folder where the template and markdown files are located during conversion.

Before the conversion process begins, a new folder is created in the project directory. This is the conversion folder, named after the current date and time. This folder can be deleted using the `clean` function, or automatically removed when converting using smart clean.

If a markdown project has an output folder defined, this is used as the conversion folder for that markdown project.

To explain more thuroughly, we need to explain the workflow of TiefDown conversion. Helpfully, there's a diagram:

As you can see, the first step of any conversion is combining the markdown files to a single Markdown file. This is done before conversion, sorting the files by chapter number. This chapter number is retrieved from the filename. The file must thus be named `Chapter X.md` where X is the chapter number. This does not need to include leading zeros. This combined file is then saved as `combined.md` in the conversion folder.

After combination, pandoc is run on the combined file to derive LaTeX, typst, epub or custom outputs in case of custom pandoc conversion.

For LaTeX and Typst templates, the output file is imported into the template file as described in the templates section. The conversion process converts the template file and stores the output in the conversion folder.

The output file is then copied to the output folder.

co
M

6 Templates

Templating in TiefDown is done in several ways:

- LaTeX templates
  - The most basic form of templating, it generates a LaTeX document from the markdown files that can be included in a LaTeX document.
  - Supports Metadata file generation.
- Typst templates
  - Similar to LaTeX, it generates a Typst document from the markdown files that can be included in a Typst document.
  - Supports Metadata file generation.
- EPUB templates
  - A legacy templating system, it generates a EPUB document from the markdown files. Convoluted and very much custom to basic usage.
  - Adds Metadata directly to the EPUB file.
  - (!) This template type should be forgone in favour of Custom Pandoc Conversion.
- Custom pandoc conversion
  - A more advanced templating system, it runs custom pandoc commands on the markdown files. This is the most flexible templating system, but also the most complex.
  - Supports Metadata insertion into command line arguments.

6.1 LaTeX Templates

LaTeX templates are the most intuitive form of templating in TiefDown, but also the most fleshed out. The basic usage generates a LaTeX document from markdown, usually output.tex, with lua-filters applied depending on the template, and then converts that template file to a PDF.

The LaTeX file must include the following:

```
\input{./output.tex}
```

This imports the converted markdown files into the LaTeX document. You may adjust the behaviour by using custom preprocessors and custom processors.

For Metadata, one can also import the metadata file, which is generated by TiefDown during the conversion process.

```
\input{./metadata.tex}
```

This file provides a macro to access metadata using the `\\meta\{\}` keyword. This can be adjusted using the metadata settings.

There are preset templates available in the core library. These give a basic framework for extension and shouldn't be taken as the only way to use LaTeX templates.

6.2 Typst Templates

Typst templates are, in concept, identical to LaTeX templates. They generate a Typst document from markdown, usually output.typ, with lua-filters applied depending on the template, and then converts that template file to a PDF.

Importing the typst file works similar:

```
#include "output.typ"
```

Again, see custom preprocessors and custom processors for more information on customization.

Metadata importing is easier as typst has an object system.

```
#import "./metadata.typ": meta
```

One can then access metadata using the `meta` object.

There are also preset templates available in the core library. Again, these are just a basic suggestion.

6.3 EPUB Templates

EPUB templates are a special version of custom pandoc conversion. They are legacy and should be avoided in favour of custom pandoc conversion. Thus, they are not documented here.

6.4 Custom Pandoc Conversion

Custom pandoc conversion is the most advanced templating system in TiefDown. It allows specifying the exact pandoc command to run on the markdown files, but does not allow any post processing like LaTeX or Typst.

A template requires a custom preprocessor to be specified. This processor defines the pandoc command to run and must include an output file. The output file must then also be included in the template for copying to the output folder.

The pandoc conversion runs in the compilation directory of the markdown project and can thus access the markdown file. This is always available as `combined.md`. See conversion folders for more information.

7 Smart Clean

Smart clean automatically removes old conversion folders. When enabled via `smart\_clean` in `manifest.toml`, TiefDown keeps only a given number of recent folders. The number is specified with `smart\_clean\_threshold` and defaults to `5`.

During conversion the library checks the amount of existing conversion folders and deletes the oldest ones once the threshold is exceeded.

8 Profiles

A profile is a named list of templates that can be converted together. Defining profiles avoids having to pass a long list of template names every time you run the converter.

Profiles are stored in the project's `manifest.toml`:

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

Use the `--profile` option with `tiefdownconverter convert` to select a profile. Markdown projects may also specify a `default\_profile`; this profile is used if none is supplied on the command line.

9 Lua Filters

Lua filters allow you to modify the document structure during Pandoc's conversion step. They are attached to templates through the `filters` field. The value may be a single Lua file or a directory containing multiple filter scripts.

Pandoc executes the filters in the order they are listed. Filters can rename headers, insert custom blocks or otherwise transform the document before it reaches the template engine.

Example filter to adjust chapter headings:

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

For more details on writing filters see the Pandoc documentation.

10 Markdown Projects

A TiefDown project can contain multiple markdown projects. Each project defines where the source files live and where the converted results should be placed. The information is stored in `[[markdown\_projects]]` entries in `manifest.toml`.

```
[[markdown_projects]]
name = "Book One"
path = "book_one/markdown"
output = "book_one/output"
```

A markdown project may define a `default\_profile` used for conversion, a list of `resources` to copy into the conversion folder and its own metadata.

10.1 Custom Resources

Resources are additional files that are copied from the markdown project directory to the conversion folder before processing. Typical examples are images, CSS files or fonts needed by a template. Specify them in the `resources` array:

```
resources = ["resources/cover.png", "resources/styles.css"]
```

10.2 Markdown Project Metadata

Project specific metadata is stored under the `metadata\_fields` table of a markdown project. These values are merged with the `[shared\_metadata]` of the project during conversion. When keys collide, the markdown project metadata overrides the shared metadata.

11 Custom Processors

Custom processors let you change the commands used during conversion. They come in two forms:

- **Preprocessors** replace the default pandoc invocation that generates the intermediate file.
- **Processors** provide additional arguments to the program that handles the template itself (for example XeLaTeX or Typst).

A preprocessor is defined under `[[custom\_processors.preprocessors]]`:

```
[[custom_processors.preprocessors]]
name = ”Enable Listings”
cli_args = [”-t”, ”latex”, ”--listings”]
combined_output = ”output.tex”
```

A preprocessor can also define a command using the `cli` field. This replaces the Pandoc preprocessing step with a custom cli command preprocessing step.

```
[[custom_processors.preprocessors]]
name = ”Copy without modification”
cli = ”cat”
cli_args = []
combined_output = ”output.tex”
```

Templates reference it with their `preprocessors` field, which also has to define a `combined\_output` field. Processors are specified similarly and referenced via the `processor` field:

```
[[custom_processors.processors]]
name = ”Typst Font Directory”
processor_args = [”--font-path”, ”fonts/”]
```

These mechanisms allow fine-grained control over the conversion pipeline when the defaults are not sufficient.