Book4matter

MARKDOWN TO PRINT, PDF, EPUB, AND HTML

Thomas Hill

This book was formatted by Book4matter, using Pandoc and Typst. The headings are set using Liberation Sans and the body with Libertinus Serif.
Produced: 20 September 2026

Contents

Introduction iv

PART ONE · GETTING STARTED 1
Installing and Building 2
Your First Book 6

PART TWO · WRITING YOUR BOOK 9
Quick Start 10

PART THREE · MARKDOWN 11
Novels and Flat Books 15
Images, Figures, and Scene Breaks 18
Tables, Quotes, Footnotes, Links and Math 20
Auto-Numbering, Cross-References, and Lists 23

PART FOUR · CONFIGURATION 27
The book_metadata.yaml File 28
The book_style.yaml File 32
Command-Line Reference 36

PART FIVE · UNDER THE HOOD 41
Default Formatting Decisions 42
How Book4matter Uses Typst 45
Customizing the Output 48
Editing the Templates 50
Importing from Word 52
Limitations 54

AFTERWORD 56

INTRODUCTION

Book4matter turns a folder of text formatted with Markdown into four outputs: a print-ready interior PDF for Amazon Kindle Direct Publishing (KDP), a digital PDF to send to readers, an EPUB for e-readers, and a standalone HTML page for the web. You write once, in plain Markdown, and the same source produces all four.

It can also format the pages for printing locally, for binding into a book.

The book you are reading is itself a book4matter project. Its chapters live in `docs/chapters/` as ordinary Markdown, its identity in `docs/book_metadata.yaml`, and its appearance in `docs/book_style.yaml`.

WHAT IT IS FOR

Book4matter is a tool used to format the books for self-publishing. You write in Markdown, simple text with a few simple options. Then you can distribute it as PDF, EPUB, or HTML, or upload the print version to Amazon KDP for printing.

It is a command-line tool, run inside Docker, and it is happiest in the hands of someone comfortable with a terminal. It is deliberately small: it does the typesetting a self-publisher actually needs and stops there.

WHAT IT DOES NOT DO

It does not design your cover. KDP accepts the print cover as a separate file, so the `print` command produces the **interior** only — the pages between the covers. The `pdf`, `epub`, and whole-book `html` outputs can

include a cover image if you supply one. Book4matter also will not write your book, fix your grammar, or upload to KDP for you.

HOW IT FITS TOGETHER

Markdown is the single source of truth. Pandoc (https://pandoc.org) parses it; for the two PDF outputs it emits a Typst (https://typst.app) fragment that a small template typesets into pages, and for EPUB and HTML it writes those formats directly. Keeping the design in a template — apart from the manuscript — means one change restyles every book, and the same words reach print, e-reader, and web without drifting apart.

The chapters that follow take you from a first build (Part One) through writing (Part Two) and configuration (Part Three) to the decisions and machinery behind the output (Part Four).

BUILDING YOUR BOOK

The commands to build your book are simple:

```
# This manual, as a 6x9 print interior PDF.
./run.sh print docs/
# The same pages as a digital PDF, with live links.
./run.sh pdf docs/
# The same book as an EPUB.
./run.sh epub docs/
# The same book as one HTML page.
./run.sh html docs/

# The same book, four pages per sheet, formatted for folding
and binding.
./run.sh impose docs/out/book4matter-interior.pdf
```

PART 1

GETTING STARTED

Three commands and a folder of Markdown stand between you and a finished book. This part installs the tool, explains the shape of a book4matter project, and walks the first build from Markdown all the way to a print-ready PDF.

1

INSTALLING AND BUILDING

Book4matter is Pandoc, Typst, Python, an EPUB validator, and a set of book fonts, working together. There are three ways to get all of that; pick whichever suits how you like to work:

1. **Pull the ready-made image from Docker Hub** — the quickest start, and the right choice for most people.
2. **Install the pieces directly** on your machine, with no Docker at all.
3. **Build the image yourself** from this repository, if you would rather not run a prebuilt one.

The first and third need Docker. The second needs no Docker but a handful of command-line tools. All three run the same `bf` commands and produce identical output.

OPTION 1: THE DOCKER HUB IMAGE

A prebuilt image is published at `book4matter/book4matter` (https://hub.docker.com/r/book4matter/book4matter). This is the fastest way to start: there is nothing to compile and no repository to clone — the tool and its templates are baked into the image, and only your book's files are read from disk.

Pull it once:

```
docker pull book4matter/book4matter
```

Then run the tool with `docker` directly. Mount the directory that holds your book at `/work`; the image does the rest:

```
docker run --rm -v "$PWD":/work book4matter/book4matter print
my-book/
```

The image's entry point is `bf`, so everything after the image name is an ordinary `bf` command. The other forms work the same way:

```
docker run --rm -v "$PWD":/work book4matter/book4matter pdf
my-book/
docker run --rm -v "$PWD":/work book4matter/book4matter epub
my-book/
docker run --rm -v "$PWD":/work book4matter/book4matter html
my-book/
docker run --rm -v "$PWD":/work book4matter/book4matter all
my-book/
```

Output lands in `my-book/out/`. If typing the full line each time gets tedious, wrap it in a shell alias or a short script of your own.

OPTION 2: INSTALL DIRECTLY, WITHOUT DOCKER

If you would rather not run Docker at all, install the tools on your machine. Book4matter needs:

- **Pandoc 3.9** — the document converter.
- **Typst 0.14** — the PDF typesetter. It bundles the Libertinus Serif body font, so there is nothing extra to install for that.
- **Python 3 with PyYAML** — the `bf` command.
- **The Liberation fonts** — Liberation Sans is used for headings.

Two more are optional: a Java runtime and EPUBCheck (https://github.com/w3c/epubcheck) if you want the EPUB validated (otherwise build EPUBs with `--no-check`), and Python's `pypdf` if you want the `impose` command.

Pandoc and Typst both ship self-contained binaries on their release pages, so the surest way to get the tested versions — on any operating system, without a package manager — is to download them directly:

- Pandoc 3.9: https://github.com/jgm/pandoc/releases
- Typst 0.14: https://github.com/typst/typst/releases

PyYAML comes from `pip` (`pip install pyyaml`), and the Liberation fonts from your operating system's font package or the project page (https://github.com/liberationfonts/liberation-fonts).

Then clone this repository — it holds the `bf` command and the templates — point `bf` at the templates, and run it as a Python module from the repository root:

```
export BF_TEMPLATES="$PWD/templates"
python3 -m bf print my-book/
```

Newer versions of Pandoc and Typst usually work, but 3.9 and 0.14 are the versions the project builds and tests against. The project's own GitHub Actions workflow, `.github/workflows/deploy-web.yml` (https://github.com/cocode/Book4matter/blob/main/.github/workflows/deploy-web.yml), downloads exactly these binaries and builds every sample book on each commit, so it is a working, always-current reference for this option.

OPTION 3: BUILD THE IMAGE YOURSELF

You can also build the Docker image from the `Dockerfile` in this repository rather than pulling the prebuilt one from Docker Hub. It is more work, but the image is assembled entirely from sources you can read — worth doing if you would rather not run a prebuilt binary you didn't build.

Clone the repository. The bundled `run.sh` builds the image the first time you use it, then runs the tool with your current directory mounted inside the container:

```
# Builds the image once, then builds the example book.
./run.sh print example/
```

That first run takes a little longer, while the image is assembled.

The templates and the `bf` command itself are baked into the image. After you change a template or update the tool, force a rebuild so the change takes effect:

```
./run.sh --rebuild print example/
```

To rebuild the image without building a book:

```
docker build -t book4matter .
```

Your book's own files — the Markdown, the YAML, the images — are *not* baked in; they are read fresh from disk on every build, so editing a chapter never needs an image rebuild.

`run.sh` only ever shells into the container. It mounts the current directory at `/work`, and — when you pass a file that lives elsewhere, such as a `.docx` in `~/Documents` — it mounts that file's folder read-only so the container can read it. Everything else on your machine is left alone.

2

YOUR FIRST BOOK

A book4matter book is just a folder. At minimum it holds two YAML files and a `chapters/` directory:

```
my-book/
  book_metadata.yaml
  book_style.yaml
  chapters/
    010-introduction.md
    020-first-steps.md
  media/
  out/
```

Only one of the two YAML files is strictly required — book4matter reads both and merges them — but keeping identity in `book_metadata.yaml` and appearance in `book_style.yaml` is the convention this manual follows. The `chapters/` directory holds one Markdown file per chapter or part, `media/` holds images referenced from chapters as `../media/...`, and `out/` is where build output lands.

CHAPTER ORDER

By default every `chapters/*.md` file is included, sorted by filename in *natural* order, so `2.md` comes before `10.md`. Numeric prefixes such as `010`, `020`, `030` make the order explicit and leave gaps to slot a new file between

two existing ones without renumbering. If you would rather order the files by hand, list them under a `chapters:` key in `book_metadata.yaml`; then only those files, in that order, are built.

If you don't need chapter titles, you can just put one chapter per file, and Book4matter will assume each file is a chapter and number them. You can still put titles on some chapters, like "# Introduction", "# Afterword".

The Novels and Flat Books chapter covers that low-structure path.

Plain `.txt` files can be used, too. Actually, Book4matter just treats .txt files just like markdown files. This could cause problems, if you have Markdown-like markup in your plain text files. But the only formatting you need in a text file is to leave a blank line between paragraphs, and put your chapters in individual files.

BUILDING EACH FORMAT

```
# For printing or KDP upload.
./run.sh print my-book/
# For sending someone a PDF.
./run.sh pdf my-book/
# For e-readers.
./run.sh epub my-book/
# For the web.
./run.sh html my-book/
```

Output always lands in the book's `out/` directory, named after the book's title: `print` writes `<title>-interior.pdf`, `pdf` writes `<title>.pdf`, `epub` writes `<title>.epub`, and `html` writes `<title>.html`.

Use `print` when you are printing a book or uploading an interior to KDP. It omits the cover and removes internal link annotations, because print platforms such as KDP expect the cover as a separate file and may reject hyperlinks inside an interior PDF. Use `pdf` when you will send someone a PDF. It can include the cover and keeps hyperlinks live for the table of contents, footnotes, and web links.

THE BUNDLED EXAMPLES

The repository ships with several examples.

- example
- novel
- docs

EXAMPLE

There is a tiny example book. Build it to confirm your setup works before pointing the tool at your own manuscript:

```
# Writes example/out/the-pocket-pipeline-interior.pdf.
./run.sh print example/
```

NOVEL

You can also look at the ./novel example, which shows simplified formatting.

DOCS

And you can also look at our documentation, in ./docs, which is also a Book4matter project.

PART 2

WRITING YOUR BOOK

A book4matter manuscript is plain Markdown. A handful of conventions turn that Markdown into a properly structured book: how headings become parts and chapters, how images are placed and sized, and how tables, quotes, and notes are set. This part covers all three.

3

QUICK START

A fast way to get started is to copy one of the provided examples, and then modify it to suit your needs.

1) Build it unmodified, in your directory.
   - Then you'll know everything works, before you start.
2) Copy your text into the ./chapters directory
3) Build your version, and see how it looks
4) Put your information into book_metadata.yaml
   - Author name, title, etc.
5) If you want, change the look of your book
   - Edit book_style.yaml

PART 3

MARKDOWN

If you haven't used it before, Markdown is a very simple way of indicating style and structure in text.

Examples:

bold

I am a heading

You can find complete documentation here (https://garrettgman.github.io/rmarkdown/authoring_pandoc_markdown.html).

Markdown heading levels are how you give a book its structure. Book4matter reads them as a hierarchy and supplies the "PART" and "CHAPTER" labels, the page breaks, and the numbering for you — you write only the titles.

Markdown	Becomes	On the page
#	**Part**	Its own divider page, an auto "PART N" label, the title set a third of the way down
##	**Chapter**	A fresh page with an auto "CHAPTER N" label and a large heading
###	Section	A bold heading, spaced from the text above it
####	Subsection	A smaller bold heading
##### / ######	Minor heading	Regular weight, still set apart from the body

NUMBERING TAKES CARE OF ITSELF

Parts are numbered one, two, three; chapters are numbered straight through, and the count does **not** restart inside each part. Both labels are generated at build time, so inserting or reordering a chapter renumbers everything automatically. Never type "Chapter 3" into a heading yourself.

FRONT MATTER

Any chapter that appears **before the first part** is treated as front matter. It carries lowercase roman folios (i, ii, iii …) and is listed in the Contents ahead of Part One; arabic page 1 begins at the first part's divider. This is where an introduction or preface belongs:

```
## Introduction {.unnumbered}
```

A book with no parts at all is simpler still: every `##` is a chapter and arabic page numbers run from page one. And a book that writes `#` for its chapters — with no parts above them — is handled too; see the Novels and Flat Books chapter for that and for numbered, title-less chapters.

TWO SMALL SWITCHES

Two optional classes ride on a heading:

- `{.unnumbered}` suppresses the auto "PART N" / "CHAPTER N" label without disturbing the count or the heading's place in the structure. (Pandoc's `{-}` shorthand means the same thing.) Use it for an Introduction or a Conclusion that should carry no number.
- `{.new-page}` forces a page break just before the heading.

TOP-LEVEL SECTIONS: AFTERWORDS AND APPENDICES

A part divider (`#`) is a grand thing: its own page, a "PART N" label, the title floated a third of the way down. Back matter — an afterword, an appendix, an acknowledgments page — wants none of that pomp, yet it still belongs at the top level of the Contents, beside the parts rather

than tucked under the last one. Add `{.section}` to a `#` heading to get exactly that:

```
# Afterword {.section}

Thanks for reading this far.
```

On the page it is set like a chapter — the title in your chapter style, with no number — and its text flows on beneath, the way a chapter opens, instead of sitting alone on a divider. In the Contents it appears at the top level with no "PART N ·" prefix. Its folios follow its position: a `{.section}` before the first part is front matter, one after the parts is main matter, and it is never itself counted as where the main matter begins.

In short: reach for `{.unnumbered}` when a heading should keep its place in the structure but lose its number, and for `{.section}` when a top-level heading should read like a chapter rather than a part.

TEXT ON A PART DIVIDER

A part is usually a page to itself with nothing but the title. If you want a few words on that divider page — an epigraph, a sentence framing the part — write them directly under the part heading, before the first chapter:

```
# Foundations

Everything in this part assumes you can already hold a frame
and keep time.

## Posture
...
```

Because the chapter files are concatenated in order, this "divider text" lives at the top of the part's own file, ahead of its first `##` chapter. It prints beneath the part title on the divider page rather than spilling onto the next.

4

NOVELS AND FLAT BOOKS

The previous chapter describes the full hierarchy — parts, then chapters, then sections. Many books never use it. A novel is the clearest case: it has chapters, often with no titles at all, just numbers, and no parts above them. Book4matter has a low-friction path for exactly this, so that a book with the least possible structure needs the least possible formatting.

THERE IS NO SUCH THING AS PARTS WITHOUT CHAPTERS

A book can have chapters and no parts, but never parts and no chapters. So when a manuscript uses no ## headings anywhere, book4matter reads # as **chapter**, not part — a plain chaptered book, arabic page numbers from page one, no divider pages. You write # for each chapter title and nothing else. This happens on its own; there is no flag to set.

`--no-parts`: WHEN YOU *DO* USE ## INSIDE A CHAPTER

Some authors naturally write # for a chapter and ## for a heading *within* that chapter. Left alone, book4matter would read those as parts and chapters. Pass `--no-parts` (or set `no-parts: true` in `book_style.yaml`)

and every heading shifts down one level: `#` becomes a chapter, `##` a section, `###` a subsection, and so on. No part dividers are produced.

```
./run.sh print my-novel/ --no-parts
```

FILES WITH NO HEADING BECOME NUMBERED CHAPTERS

Here is the smallest possible book: a folder of text files with no markup in them at all.

```
my-novel/
  book_metadata.yaml
  chapters/
    01.txt
    02.txt
    03.txt
```

Each file with no heading becomes one chapter, numbered in the order the files are read — the first file is Chapter 1, the next Chapter 2, and so on. You never type the numbers; book4matter supplies them, so inserting or reordering a file renumbers the rest. The chapter opens with a centered numeral over a short rule, the classic novel look.

Because nothing in the book carries a title, there is nothing to list, so the **Contents page is left out** automatically. (Set `toc: true` in `book_style.yaml` if you want one anyway; the numbered chapters then appear as "Chapter 1", "Chapter 2", and so on.)

PLAIN TEXT FILES

Chapters may be `.txt` as well as `.md`. A `.txt` file is still read as Markdown — so blank lines separate paragraphs, `---` becomes an em dash, and a line of `* * *` becomes a scene break — but a novelist writing ordinary prose need not know any of that. Type paragraphs, leave a blank line between them, and it comes out right.

A NAMED SECTION AMONG NUMBERED CHAPTERS

Most novels want at least a titled page or two — a prologue, an introduction, an afterword. Give that one file a `#` heading and leave the rest without one:

```
chapters/
  00-introduction.txt      # begins with "# Introduction"
  01.txt                   # no heading
  02.txt                   # no heading
  99-afterword.txt         # begins with "# Afterword"
```

A titled file set among untitled ones is treated as a **named section**: it shows its title (Introduction, Afterword) and is **not** given a chapter number, nor does it disturb the count. The untitled files remain Chapter 1, Chapter 2. Because the book now has titles, a Contents page is included, listing the named sections alongside the numbered chapters.

If instead *every* file has a `#` title, they are all ordinary titled chapters, numbered straight through — the titled-chapter novel, or any chaptered non-fiction book built with `--no-parts`.

THE BUNDLED NOVEL

The repository ships a second example, `novel/`, alongside the `example/` book. It is a very short fable with no titles, no parts, and no markup — just numbered `.txt` chapters — so you can see the whole of this chapter at work:

```
# Writes novel/out/the-smallest-fire-interior.pdf.
./run.sh print novel/
```

5

IMAGES, FIGURES, AND SCENE BREAKS

WHERE IMAGES LIVE

Keep images in a `media/` folder at the root of your book. Chapter files sit one level down, in `chapters/`, so they reference an image with a leading `../`:

```
![A frame from the Vienna Ball](../media/vienna.jpg)
```

That one path works everywhere — print, digital PDF, EPUB, HTML, and your Markdown editor's preview — because the build compiles from the book root and never rewrites your paths.

CAPTIONS AND CENTERING

A captioned image becomes a centered figure automatically, in every format:

```
![The closed hold, seen from above](../media/hold.png)
```

An image with **no** caption is treated as inline text and sits flush left. To center a caption-less image on its own line, tag it `{.center}`:

```
![](../media/ornament.svg){.center}
```

SIZING

Set the size with `width` or `height`:

```
![](../media/diagram.svg){width="3in"}
```

Give **one** dimension and the other scales to keep the image's proportions. Setting both `width` and `height` to a ratio that does not match the image can crop it, so as a rule size by a single dimension. SVG line art is welcome and scales without loss; for photographs, supply roughly 300 DPI at the printed size, which is what KDP expects for a sharp result.

WRAPPING TEXT AROUND AN IMAGE

To float an image to one side and let the section's text flow beside it, tag it `{.wrap-right}` or `{.wrap-left}`:

```
![](../media/step.svg){.wrap-right width="2in"}

The text of this section flows up the side of the image and
returns to full
width once it clears the bottom edge.
```

The wrap runs from the image to the end of its section — that is, until the next heading at the same level or shallower. In the PDF outputs this is a true text wrap; in EPUB and HTML it is a CSS float, which modern e-readers honor and older ones quietly ignore (the image just becomes a normal block — fine either way).

SCENE BREAKS

A "scene break" between passages is written as three hyphens alone on their own line. In EPUB and HTML it renders as a centered * * * ornament instead of a hard rule.

Print note. *The PDF pipeline does not yet render a bare scene break — the Typst template defines no rule for it — so avoid one in a book you build to print until that support is added.*

6

TABLES, QUOTES, FOOTNOTES, LINKS AND MATH

Ordinary book prose — tables, quotations, notes, and links — needs no special syntax beyond standard Markdown. Book4matter sets each in book style.

TABLES

Write a normal Markdown pipe table and align a column with colons in the divider row. In the PDF outputs it is set in book (booktabs) style: a rule above, a rule under the header, and a rule below — no vertical lines, no shaded cells — with the header in bold and figures set in tabular (fixed-width) columns so digits line up.

```
| Step | Count | Beats |
|------|------:|------:|
| Rise |     1 |     1 |
| Turn |     2 |    23 |
| Fall |     3 |   456 |
```

BLOCK QUOTES

Mark a quotation with >; never indent by hand. A block quote is set a little smaller and in italic, indented from both margins with air above and below:

Waltzing has undergone so many alterations that the dance of today bears little resemblance to its ancestor — yet the family likeness survives.

FOOTNOTES

Markdown footnotes work as usual. In the two PDF outputs they fall to the foot of the page; in EPUB and HTML they collect at the end of the chapter, linked from a superscript marker. The digital `pdf` output keeps its footnote markers clickable; the `print` output does not, because KDP rejects internal hyperlinks in print interiors.

While you can put footnotes anywhere, we recommend putting them in the file where they are referenced. If you put all the footnotes in an otherwise empty file, you may end up with a "Ghost Chapter" in the table of contents, represeting the empty file. This will happen with an empty file, too.

```
Practice the rise slowly at first.[^rise]

[^rise]: Rising too early is the commonest beginner's
mistake.
```

LINKS

Links survive to every format, but the target decides whether they are clickable. Use the digital `pdf`, EPUB, and HTML outputs when links should work on screen; use `print` when the file is going to a printer or to KDP.

- **In digital PDF, EPUB, and HTML** links stay live. That includes the table of contents, footnotes, cross-references, and external web links.

- **In print** there are no clickable annotations, because KDP rejects them, but the links still guide the reader. An internal cross-reference to another part of the book — `[Rotation](#rotation)` — prints as its text followed by the page it points to, so `"Rotation" (page 42)`; the page number is resolved for you at build time and updates automatically as the book repaginates. (A link whose target is missing simply prints its text.) External web links print the destination after the visible text, so `[CLICK HERE](https://www.example.com)` becomes `CLICK HERE (https://www.example.com)`. A bare autolink shows its URL once. Email addresses are kept whole and never broken at a hyphen across a line.

Write internal links with Markdown link syntax, `[text](#anchor)`, where the anchor matches a heading. Raw HTML anchors like `<a href="#anchor">` do not survive to print — only the Markdown form is turned into a page reference.

PUNCTUATION

Write punctuation the Markdown way and the build sets it properly: `--` becomes an en dash, `---` an em dash, and straight quotes turn into curly ones. There is no need to paste Unicode dashes or smart quotes by hand.

MATH

Unfortunately, we don't yet have math expressions working fully in HTML and EPUB. They work fine in pdf and print.

7

AUTO-NUMBERING, CROSS-REFERENCES, AND LISTS

Books are full of things that are numbered and referred to: exercises, figures, charts, tables. Numbering them by hand is a trap — insert one new figure early on and every later number, and every reference to it, is suddenly wrong. Book4matter numbers these *tracked items* for you, lets you refer to them by a name you choose (never by number), and can gather them into a list — a *List of Exercises*, a *List of Figures* — with the correct page beside each.

REGISTER THE KINDS YOU USE

First tell the book which kinds of thing you track, in `book_metadata.yaml`. Each kind gets a `label`: the word printed before the number and heading its list.

```
tracked:
  exercise: { label: "Exercise" }
  figure:   { label: "Figure" }
  chart:    { label: "Chart" }
  table:    { label: "Table" }
```

Registering the kinds up front is what keeps the feature safe: only these words are treated as tracked-item markup, so an ordinary `{ratio:`

`3:2}` written in your prose is left completely alone. A kind is lowercase letters, digits, or hyphens.

DEFINE AN ITEM WHERE IT BELONGS

Give each item a short id — a name that means something to you, unique within the whole book. Two ways to write a definition:

```
{exercise:pushups:def}
{exercise:pushups Push-ups to failure}
```

The first defines an exercise with the id `pushups` and no title. The second does the same but adds a title: **anything after the first space is the title**. Both print in place as *Exercise 3* (whatever the next number is); the title is not printed here — it is saved for the list at the back. Write any caption text around the token yourself:

```
{figure:rotary A rotary telephone}. *An early rotary dial, c.
1950.*
```

For a figure or chart, put the definition in the caption beside the image; for an exercise, at the head of the exercise. Each kind counts on its own, so your first figure is *Figure 1* even if three exercises came before it.

REFER TO AN ITEM BY NAME

Anywhere else — before or after the definition — refer to the item with just its kind and id, no `:def`:

```
Warm up before you attempt {exercise:pushups}.
```

That prints *Exercise 3*, and the number is always right because the build looks it up. What the reference becomes depends on the output, exactly like any other internal link:

- **In digital PDF, EPUB, and HTML** it is a live, clickable link to the item.
- **In print** there are no clickable links (KDP forbids them), so it prints the number followed by the page, as *Exercise 3 (page 42)* — resolved for you and kept correct as the book repaginates.

Because you never type a number, inserting, removing, or reordering items is free: every number and every page updates on the next build.

LIST THEM AT THE BACK (OR THE FRONT)

Drop this on a line of its own wherever you want the list to appear:

```
{index:exercise}
```

It prints every exercise in order of appearance, each one a link to the item. In the print and digital PDF outputs each entry carries its **page number**, like a table of contents for your exercises; in EPUB and HTML, which have no fixed pages, the entry is simply a tappable link. Give it a heading of its own:

```
# List of Exercises {.section}

{index:exercise}
```

By book convention a *List of Figures* or *List of Tables* sits in the front matter, just after the table of contents, while an alphabetical index of terms goes in the back. Book4matter does not force either: the list appears wherever you place `{index:...}`, and the page numbers resolve correctly whether the content it points to comes before or after it.

WHEN SOMETHING IS WRONG, THE BUILD SAYS SO

The system fails loudly rather than printing a quietly wrong book:

- A reference to an id that is **never defined** stops the build and names it — usually a typo in the id, or a definition you forgot to mark with `:def`.
- **Defining the same id twice** for one kind stops the build.
- An **unregistered kind in an unmistakable token** — `{exercise:x:def}` or `{index:exercise}` when `exercise` isn't in your `tracked:` map — stops the build. This is what catches the commonest mistake: forgetting the `tracked:` block altogether, or mistyping a kind (`{index:figurse}`).

The two *unmistakable* forms are the `:def` keyword and `{index:...}`; they can only be tracked markup, so an unregistered kind there is always an error. The plainer forms overlap with ordinary writing, so `{exercise:pushups}` or `{exercise:pushups A title}` with an *unregistered* kind is left as text and prints verbatim (you will see it and can fix it) rather than being seized from your prose. Either way, tokens inside `code spans` and fenced code blocks are never touched — which is how this chapter shows them to you.

Numbering is book-wide, so the one output that cannot do it is a **single-chapter HTML fragment** (`html chapter N`): with only one chapter in hand there is no way to number across the book or resolve a reference to another chapter, so tracked tokens are left as-is there. Every whole-book output — print, PDF, EPUB, and the whole-book HTML page — handles them fully.

PART 4

CONFIGURATION

Two YAML files and a handful of command-line flags control everything about a book: what it is, how it looks, and how it is built. This part is the reference — every key in `book_metadata.yaml` and `book_style.yaml`, then every sub-command of the tool.

8

THE BOOK_METADATA.YAML FILE

`book_metadata.yaml` holds the book's identity — the facts that stay true no matter how the book is styled. Every key is optional except `title`. Anything you leave out (or set to an empty string) is simply omitted: no blank author, no empty copyright line.

`title` — The book's title; defaults to "Untitled". A `\n` in the value stacks the title across lines on the print title page — `"The\nRotary\nWaltz"` sets three lines — while everywhere else (PDF metadata, EPUB, HTML, running heads) the breaks collapse to spaces.

`subtitle` — A subtitle, set beneath the title on the title page and carried into the digital outputs. Turn on `title-rule` (in `book_style.yaml`) to draw a rule between the two on the PDF title page.

`author` (or **`authors`**) — One name as a string, or several as a YAML list. Either key works. The author appears on the title page, in the PDF and EPUB metadata, and in the "Also by" heading.

```
author: "Thomas Hill"
# Or:
authors:
  - "Ada Lovelace"
  - "Charles Babbage"
```

`year` — The publication year. Used as the EPUB date (`<dc:date>`).

`publisher` — Publisher name, shown on the copyright page and in the EPUB metadata.

`rights` — The copyright or rights statement printed on the copyright page (and stored as the EPUB `<dc:rights>`). When you pass `--build-id`, a "Printing: …" line is appended to it in the EPUB.

`isbn` — Printed on the copyright page and used as the EPUB identifier (tagged as ISBN-13).

`credits` — An acknowledgements line on the copyright page — a cover credit, for instance: `"Cover photo courtesy of RJ Muna."`

`language` — A language code such as `en` or `fr`. It sets the document language for the PDF and the EPUB (`<dc:language>`) and drives hyphenation. Defaults to `en`.

`cover` — A path (relative to the book directory) to a cover image. The `pdf` command places it as the first page, `epub` embeds it as the EPUB cover, and the whole-book `html` output embeds it near the top of the page. `print` ignores it: KDP takes the print cover as a separate upload, so it never belongs in the interior PDF. Builds that use the cover stop if the file is missing.

`html-body-end` — A path (relative to the book directory) to a file whose contents are injected *verbatim* just before `</body>` in the whole-book `html` output. This is the clean place for an end-of-body script — a web-analytics snippet, for instance — without editing a chapter. It is HTML only by design: `epub`, `pdf`, and `print` never read this key, so they stay script-free (and the EPUB stays valid: EPUB restricts scripting, so keeping the snippet out of it is what you want). The file is passed through as-is — raw HTML and JavaScript — so put a complete `<script>…</script>` there. If the file is missing, the build warns and produces the page normally, just without the snippet — a mistyped name costs you the snippet, never the page's formatting.

```
# book_metadata.yaml
html-body-end: analytics.html
```

THE “ALSO BY” PAGE

Give an `also-by:` list and book4matter adds an “Also by *Author*” page in the front matter of the PDF outputs; omit it and there is no such page. Each entry needs a `title` and may add a `cover` image, a `qr` image, and a `url`. A bare string is treated as a title-only entry.

```
also-by:
  - title: "How to Teach Ballroom Dancing"
    cover: "media/htbd-cover.jpg"
    qr: "media/htbd-qr.png"
    url: "howtoteachballroomdancing.com"
  # Title only.
  - "An Earlier Work"
```

TRACKED ITEMS

`tracked:` registers the kinds of thing book4matter auto-numbers, cross-references, and can list — exercises, figures, charts, tables, whatever you use. Each kind maps to a `label`, the word printed before the number (*Exercise 3*) and heading its list. A bare string is shorthand for the label.

```
tracked:
  exercise: { label: "Exercise" }
  figure:   { label: "Figure" }
  chart: "Chart"
```

Omit the key and nothing is tracked. How to write the `{...}` tokens themselves — defining items, referring to them, and placing a list — is covered in the *Auto-Numbering, Cross-References, and Lists* chapter.

LISTING CHAPTERS EXPLICITLY

By default every file in `chapters/` is built, in natural filename order. To fix the order by hand instead, list the files under `chapters:`, relative to the book directory; only those files, in that order, are built. A listed file that does not exist stops the build.

```
chapters:
  - chapters/010-introduction.md
  - chapters/020-first-steps.md
```

Not a metadata key: *`--build-id` is a* command-line *flag, not something you put here, because it changes from build to build. See the Command-Line chapter.*

9

THE BOOK_STYLE.YAML FILE

`book_style.yaml` controls appearance only — page size, margins, fonts, and a little page furniture. Point several books at one style file and they share a look; restyle them all by editing that one file. Every key has a sensible default, so the smallest possible style file is no file at all.

`trim` — The finished page size. Write it as `6x9` (inches), or as an explicit mapping for an unusual size. Defaults to `6x9`, the standard US trade paperback.

```
trim: 6x9
# --- or ---
trim: {width: 5.5, height: 8.5}
```

`margins` — The four page margins in inches: `inside`, `outside`, `top`, `bottom`. The **inside** margin is the binding (gutter) edge and is normally the widest, because the binding swallows part of every page. Omit `margins` and book4matter fills them in: `outside` 0.625, `top` and `bottom` 0.75, and an inside gutter sized to the book's length. With no `--pages` hint it uses a generous 0.875″ (safe to ~500 pages); with `--pages N` it picks KDP's minimum for that length plus 0.125″ of breathing room:

Interior pages	KDP minimum gutter	With `--pages`
≤ 150	0.375″	0.5″
151–300	0.5″	0.625″
301–500	0.625″	0.75″
501–700	0.75″	0.875″
701+	0.875″	0.875″

Anything you set here overrides those defaults.

`font` — The body typeface. Defaults to **Libertinus Serif**, which is bundled in the image, so the default needs no setup. To use another face, drop its files into the book's `fonts/` folder and name the family here; see the Customizing chapter.

`heading-font` — The face for parts, chapters, and headings. Defaults to **Liberation Sans** (a Helvetica-metric face, also bundled).

`font-size` — Body text size, for example `11pt`. A bare number is read as points (`11` means `11pt`). Defaults to `11pt`.

`blockquote` — The shared appearance of ordinary Markdown blockquotes (`>`). `width` controls the quote measure relative to the normal text width; `rule-width` controls each centered rule above and below it; `spacing` is the vertical air around the quote; and `text-align` and `font-style` control the quote text. The defaults are `80%`, `50%`, `1.25em`, `center`, and `italic`. The same map is used by the PDF, EPUB, and HTML outputs.

`indent` — *PDF outputs only.* How paragraphs are indented. `all` (the default) indents every paragraph's first line, openers included; `standard` leaves the first paragraph after a heading flush and indents the rest (the trade-book convention, and what EPUB and HTML already do); `none` drops indents entirely and tells paragraphs apart by the space between them. Defaults to `all`.

`toc` — Whether to include the Contents page in the PDF outputs. Defaults to `true` for a book that has any titles, and to `false` for a wholly un-titled novel (nothing to list). Set it explicitly to force the issue either way. The Contents lists parts and chapters; a front-matter chapter shows

its roman folio. EPUB and whole-book HTML always get a generated navigation structure from Pandoc.

`no-parts` — *PDF outputs only.* When `true`, treat top-level `#` headings as chapters rather than parts, shifting every heading down one level (`#` → chapter, `##` → section, …). Use it for a book that has chapters but no parts and writes `#` for the chapter. Defaults to `false`, though a book that uses no `##` anywhere is read this way regardless (see the Novels and Flat Books chapter). The `--no-parts` command-line flag does the same thing.

`toc-depth` — How many heading levels the Contents includes. `2` (the default) lists parts and chapters; `3` also lists topics (level-3 `###` headings), set smaller and nested under their chapter. Applies to print, digital PDF, EPUB, and HTML.

`running-heads` — *PDF outputs only.* When `true`, body pages carry a small-caps running head: the book title on left-hand (verso) pages and the current chapter on right-hand (recto) pages. It is suppressed in the front matter and on any page where a part or chapter opens. Defaults to `false`. EPUB and HTML ignore it — those readers paginate themselves.

`title-rule` — When `true`, draw a horizontal rule between the title and subtitle on the title page. Defaults to `false`.

`chapter-style` — *PDF outputs only.* The chapter opener's layout. `centered` (the default) centers a bare chapter numeral over a short rule with the title centered beneath — which carries a wrapped title gracefully. `left` instead prints a "CHAPTER N" label with the title flush left. Defaults to `centered`.

`part-style` — *PDF outputs only.* The part divider's layout. `classic` (the default) prints an all-caps label and arabic number ("PART 1") above the title in bold capitals, sitting a third of the way down the page. `fancy` instead sets the divider in the body serif as a small letterspaced label, a large Roman numeral, a short rule, and the title in title-case — the literary look of a trade book's section openers. Any text written beneath the part heading in its chapter file prints as a centered italic blurb on the divider itself. Defaults to `classic`.

`part-label` — The word for the top-level division, replacing "Part" everywhere it is auto-generated: the PDF divider and table of contents (set in capitals there, so `SECTION TWO · ...`), and the EPUB and HTML labels. Defaults to `Part`. Set it to `Section`, `Book`, `Volume`, or anything else. It does not touch wording you write yourself in a heading or in body text.

`parts-recto` — *PDF outputs only.* When `true`, every part divider begins on a recto (the right-hand, odd-numbered page); a blank page is inserted before it when the previous page would otherwise leave it on a verso. The blank is counted and may carry a folio. Defaults to `false`. EPUB and HTML have no fixed page sides, so they ignore it.

`chapters-recto` — *PDF outputs only.* The same for chapters: when `true`, each chapter opens on a recto page. Defaults to `false`.

Not supported: *a `line-height:` key appears in some older example files, but the tool does not read it — body leading is fixed in the template. Leave it out.*

10

COMMAND-LINE REFERENCE

Every command runs through `run.sh`, which builds the Docker image if it is missing and then runs the tool with your current directory mounted inside the container. The general form is:

```
./run.sh [--rebuild] <command> [arguments]
```

`--rebuild` forces the image to be rebuilt first — do this after changing a template or updating the tool. The commands are `print`, `pdf`, `epub`, `html`, `import`, and `impose`. For the book-building commands, `bookdir` defaults to the current directory.

PRINT

```
./run.sh print [bookdir] [--pages N] [--keep] [--no-parts]
[--build-id ID]
```

Builds the print interior PDF to `bookdir/out/<title>-interior.pdf`. Use this for KDP upload or for a file you intend to print as a book. The print interior does not include the cover, because KDP takes the print cover as a separate upload. It also removes internal hyperlinks from the table of contents, footnotes, and cross-references, because Amazon KDP enforces no hyperlinks in a print interior. External web links are converted to visible text plus their destination, as in `CLICK HERE (https://www.example.com)`.

- `--pages N` — the estimated final page count. It selects a KDP-appropriate inside gutter (see the margins table in the previous chapter). Omit it and a generous default gutter is used. Any `margins` in `book_style.yaml` win regardless.
- `--keep` — keep the intermediate Typst files (`_body.typ`, `_meta.typ`, `book.typ`, `main.typ`) in `out/` instead of deleting them. Handy when you want to read the generated Typst.
- `--no-parts` — treat `#` headings as chapters, not parts, shifting every heading down one level. See the Novels and Flat Books chapter.
- `--build-id ID` — a printing identifier, printed as "Printing: *ID*" on the copyright page.

PDF

```
./run.sh pdf [bookdir] [--pages N] [--keep] [--no-parts] [--
build-id ID]
```

Builds a digital PDF to `bookdir/out/<title>.pdf`. Use this when you will send someone a PDF directly. It uses the same page layout as `print`, but keeps hyperlinks live for the table of contents, footnotes, cross-references, and external links. If `book_metadata.yaml` has a `cover:` image, the digital PDF includes it as the first page.

- `--pages N` — the estimated final page count, using the same gutter logic as `print`.
- `--keep` — keep the intermediate Typst files in `out/`.
- `--no-parts` — treat `#` headings as chapters, not parts (as for `print`).
- `--build-id ID` — a printing identifier, printed as "Printing: *ID*" on the copyright page.

EPUB

```
./run.sh epub [bookdir] [--no-check] [--build-id ID]
```

Builds an EPUB 3 to `bookdir/out/<title>.epub`. The result is validated with epubcheck — the same kind of check KDP runs on upload — and validation errors fail the build.

- `--no-check` — skip the epubcheck step.

- `--build-id ID` — appended to the rights line in the EPUB metadata.

The EPUB cover, if any, comes from the `cover:` key in `book_metadata.yaml`.

HTML

```
# Whole book: one standalone page with a clickable table of
contents.
./run.sh html [bookdir]
# Just the contents, as a link-free fragment.
./run.sh html toc [bookdir]
# One chapter, as an HTML fragment.
./run.sh html chapter NN [bookdir]
```

- The whole-book form is a single self-contained page (resources embedded) with a clickable table of contents — the form used for the book4matter website. It links an optional `book_style.css` after the embedded styles, so dropping a `book_style.css` next to the page (or copying your project's CSS into `out/` under that name) themes the site; omit it and the embedded styling stands. If `book_metadata.yaml` has a `cover:` image, the whole-book page embeds it near the top.
- `toc` emits a `<nav class="book-toc">` of plain, unlinked titles to drop into a web page (the in-book anchors would point nowhere off-site).
- `chapter NN` picks one chapter: `NN` matches the leading number in a filename (`010-foo.md` is `10`), or a 1-based position in the chapter list, or a filename/stem.
- `--build-id` is accepted for parity with the other commands but is not stamped on web output.

IMPOSE

```
./run.sh impose <file.pdf> [--paper P] [--signature N] \
  [--single] [--margin INCHES] [-o NAME]
```

Rearranges a PDF two-up into printable signatures for home binding. It is separate from the `print` command: KDP and commercial print shops impose pages for you, but `impose` is useful when you are printing on a home printer, folding the sheets, and binding the book yourself. A *signa-*

ture is a small stack of sheets folded together down the middle; several signatures are then stacked and sewn or glued into a codex.

`impose` takes any PDF, not just a book4matter interior. The output is written next to its input, so `docs/out/book4matter-interior.pdf` produces `docs/out/book4matter-signatures.pdf`. Print the result double-sided, flipping on the short edge, then fold each signature down the middle. The page streams are Flate-compressed, so the imposed PDF stays about the size of the input.

- `--paper letter` or `--paper a4` — the sheet you print on. The default is `letter`.
- `--signature N` — sheets per signature. The default is `4`, which makes 16-page signatures because each sheet folds to four pages.
- `--single` — impose the whole PDF as one folded booklet, useful for thin saddle-stitched work.
- `--margin INCHES` — safe margin inside each half-page cell. The default is `0.25`.
- `-o NAME` or `--output NAME` — output filename, still written beside the input.

IMPORT

```
./run.sh import <file.docx> [bookdir] [--no-split] [--force]
```

Converts a Word manuscript into `chapters/*.md` plus an extracted `media/` folder, splitting the document into one file per Heading 1.

- `--no-split` — write a single Markdown file instead of splitting on H1.
- `--force` — overwrite a `chapters/` directory that already has files.

The Importing from Word chapter covers this in full.

STAMPING BUILDS WITH –BUILD-ID

`--build-id` is how you make a given printing identifiable. Compute the value once — usually the repository's short git hash — and pass it to the outputs that stamp it. This manual's own `build.sh` does exactly that:

```
BUILD_ID="$(git rev-parse --short=10 HEAD)"
./run.sh print docs/ --build-id "$BUILD_ID"
./run.sh pdf docs/ --build-id "$BUILD_ID"
./run.sh epub docs/ --build-id "$BUILD_ID"
```

PART 5

UNDER THE HOOD

You can publish a book without reading any further. This part is for the curious — and for anyone who wants to bend the output to their own taste. It covers the formatting decisions baked into the default look, how Pandoc and Typst actually turn Markdown into a page, how to restyle a book, and how far the templates can be pushed.

11

DEFAULT FORMATTING DECISIONS

The default look is a set of deliberate choices about how a book should read. You can change any of them — the next chapters show how — but it is worth knowing what they are and why.

BODY TEXT

The body is set justified and hyphenated, so the right margin stays even, with lining (uniform-height) figures rather than old-style ones. The default face is Libertinus Serif at 11pt: a quiet, readable book face bundled in the image, so the default needs no font setup and nothing is fetched from the network.

PARAGRAPH INDENTS

Paragraphs are set off by a first-line indent rather than by blank space — the book convention. In the PDF outputs, `indent:` in `book_style.yaml` chooses among three styles:

- `all` (the default) — indent *every* paragraph, including the one that opens a chapter or section. The house style: no flush-left openers.
- `standard` — leave the opening paragraph after a heading flush and indent the rest. The classic trade-book look, and what EPUB and HTML already do, so `standard` makes print match them.

- `none` — no indents at all; paragraphs are told apart by the space between them (block style).

EPUB and HTML always use the `standard` look — readers expect it on screen and can override it anyway — so `indent:` is a PDF setting.

HEADINGS

Headings are set in a sans-serif face (Liberation Sans by default), in capitals, ragged-right: they never justify or hyphenate, so a long title wraps whole words instead of stretching or splitting. Each level has its own job — a part is a divider page, a chapter opens a fresh page, and the lower levels are run-in section headings.

PAGE NUMBERS AND RUNNING HEADS

Front matter carries lowercase roman folios (i, ii, iii …); arabic numbering begins at the first part. Display pages — the title, copyright, and "Also by" pages, and every part divider — show no folio, though they still count, so the numbering downstream stays correct. Running heads are off by default; when on, they appear only on ordinary body pages (never where a part or chapter opens), and only in the PDF outputs.

PARTS AND CHAPTERS

A part divider sets its title about a third of the way down the page, rather than dead centre, with room beneath for optional divider text. Chapters open on a fresh page with a centered numeral over a short rule and the title centered beneath (set `chapter-style: left` for a flush-left "CHAPTER N" opener instead). Both are numbered automatically, and chapter numbers run straight through the whole book instead of resetting inside each part.

Two more `book_style.yaml` switches, `parts-recto` and `chapters-recto`, make parts or chapters begin on a recto (right-hand) page, inserting a blank page when needed — the traditional way to open a

chapter in a well-set book. The inserted blank is counted in the page numbering (it may show a folio), and the main matter still starts at page 1.

FIGURES

A captioned image becomes a centered figure; figures are *not* auto-numbered ("Figure 1.2"), because trade books usually refer to an image in prose rather than by number. How tables, quotes, footnotes, and links are set is described where you write them, in the *Tables, Quotes, Footnotes, and Links* chapter.

LINKS IN PDF AND PRINT

The `pdf` command is for screen reading, so hyperlinks stay live there: the table of contents, footnotes, cross-references, and external links are all clickable. The `print` command is for physical books and KDP interiors, which forbid clickable annotations — so instead of dropping the reader, a print cross-reference to another part of the book prints as its text plus the target page, `"Rotation" (page 42)`, with the page number resolved at build time. External web links print their destination in parentheses after the visible text, and email addresses are kept whole so they never break at a hyphen. EPUB and HTML links stay live and bare.

12

HOW BOOK4MATTER USES TYPST

TWO ROUTES FROM ONE SOURCE

Markdown is the single source. Pandoc parses it, and from there the formats diverge:

- **Print and digital PDF** take the long way round. Pandoc converts the Markdown into a Typst *body fragment*, and Typst typesets that into pages.
- **EPUB and HTML** are written by Pandoc directly.

Why route PDF output through Typst at all? Because Typst alone cannot produce EPUB, and authoring straight in Typst would forfeit the EPUB and HTML outputs. With Pandoc in the middle, all four outputs come from the same Markdown. Typst earns its place on the page-layout side: it is fast, needs no enormous TeX installation, embeds fonts automatically, and ships in the same `pandoc/typst` image.

WHAT A PDF BUILD GENERATES

Build with `--keep` and you can read the intermediate Typst the pipeline writes into `out/`:

- **`_meta.typ`** — a Typst dictionary built from your `book_metadata.yaml` and `book_style.yaml`: title, authors, trim, margins, fonts, and the rest.
- **`_body.typ`** — Pandoc's Typst rendering of your chapters, with a short prelude of imports prepended.
- **`book.typ`** — the template, copied in from the image.
- **`main.typ`** — the short entry point that ties them together:

```
#import "book.typ": book
#import "_meta.typ": meta
#show: book.with(meta)
#include "_body.typ"
```

Typst compiles `main.typ` into either the print interior PDF or the digital PDF. Without `--keep`, these files are removed after the build.

The two commands share this pipeline, then differ in the metadata passed to the template. `print` removes internal hyperlinks and omits the cover; `pdf` keeps links live and includes the configured cover image as the first page.

THE TEMPLATE

`book.typ` owns the PDF page geometry — the exact trim, the asymmetric binding gutter used by print books, font embedding, and page numbering — and the styling: the title and copyright pages, the table of contents, and the show rules that turn each heading level into a part, a chapter, or a section. That styling is adapted from the MIT-0-licensed *ilm* Typst template.

THE PANDOC FILTERS

Small Lua filters bridge Markdown and each output so the same source behaves consistently across formats:

- **PDF outputs:** `parts.lua` numbers the parts, acts on the `{.unnumbered}`, `{.new-page}`, and `{.section}` classes, switches the front matter over to the main-matter page numbers, and lifts part-divider text onto the divider page; `wrap.lua` turns a `.wrap-left` / `.wrap-right` image into a Typst text wrap and centers a `.center` image.

- **EPUB and HTML:** `epub-parts.lua` injects the “Part N” / “Chapter N” labels, `epub-wrap.lua` floats wrap images with CSS, and `toc-list.lua` builds the link-free contents used by `html toc`.

FONTS

Typst embeds whatever fonts a build uses. The defaults (Libertinus Serif and Liberation Sans) live in the image. For a custom face, Typst also searches a per-book `fonts/` folder — the subject of the next chapter — so the fonts travel with the book and nothing is installed or downloaded. The `wrap-it` package behind text wrapping is vendored into the image as well, so a compile never has to reach the network.

13

CUSTOMIZING THE OUTPUT

Most of what you will want to change needs no template editing at all — it is a line in `book_style.yaml`. This chapter covers the customization that is built in; the next goes further, into the templates themselves.

RESTYLING THROUGH BOOK_STYLE.YAML

Trim size, margins, body and heading fonts, type size, the Contents page, running heads, the title rule — all of these are keys in `book_style.yaml` (see the reference in Part Three). Because appearance lives in that one file, you can copy or symlink a single `book_style.yaml` across several books to give them a shared house style, and restyle them all by editing it in one place. The manuscript and its metadata never change.

USING A CUSTOM FONT

The default faces are bundled, but you can use any font you have licensed:

1. Make a `fonts/` folder at the root of your book.
2. Drop the `.ttf` or `.otf` files into it.

3. Name the family in `book_style.yaml`:

```
font: "EB Garamond"
heading-font: "Cormorant"
```

When a `fonts/` folder is present, book4matter points Typst at it automatically, so the fonts travel with the book project — no system install, no image rebuild. System and bundled fonts are still searched, so a missing `fonts/` folder is fine. The PDF builds embed whatever is used, so the font's license must permit PDF embedding.

True to this project's grain, prefer a real, licensed book face that you keep with the project over pulling one from a web font service. It keeps the build self-contained and the typography under your control.

FONTS IN EPUB AND HTML

EPUB and HTML deliberately do *not* embed your custom fonts. `epub.css` names only the generic `serif` and `sans-serif` families and lets the reader's device choose — which is what readers expect, and what lets them override fonts anyway. A custom `font:` therefore affects the two PDF outputs only.

14

EDITING THE TEMPLATES

Beyond fonts and the `book_style.yaml` knobs, the look is governed by the templates baked into the Docker image. You can change them, with one thing understood up front: **there is no theme or plugin system.** Customizing the templates means editing the project's own template files and rebuilding the image. The change then applies to every book you build with that image, so this is the path for shaping *your* house style, not for per-book themes.

THE FILES

- **`templates/book.typ`** — the PDF page template. It holds the page geometry, the title / copyright / "Also by" / contents pages, and the show rules that style every heading level. Most PDF-appearance changes happen here: heading sizes and spacing, the title-page layout, the running-head format, the table and block-quote styling.
- **`templates/epub.css`** — the styling for EPUB *and* the HTML outputs. Edit this to change how the e-book and the default web page look.
- **`templates/*.lua`** — the Pandoc filters from the previous chapter. Edit these only to change structural behavior: how labels are injected, how wraps work, how the contents list is built.

APPLYING A CHANGE

The templates live inside the image, so after editing one, rebuild:

```
./run.sh --rebuild print mybook/
```

Until you rebuild, the old templates are what run.

WHAT BOOK.TYP EXPECTS

`book.typ` is driven by a `meta` dictionary that the tool assembles from your YAML (you can read it as `_meta.typ` in a `--keep` build). The template consumes a fixed set of keys — the ones documented in Part Three. Changing how an existing value is *used* is a `book.typ` edit on its own. Introducing a brand-new setting would also mean teaching the tool to pass it through, which is a code change beyond the scope of styling.

RESTYLING THE WEBSITE WITHOUT TOUCHING THE TEMPLATE

The HTML output is styled by `epub.css`, but you need not edit that shared file to restyle the *site*. Every page that `bf html` generates links an optional `book_style.css` *after* the built-in styles, so any rules in that file win on the cascade. The file is optional: drop a `book_style.css` next to the generated page — or have your build script copy one into `out/` — and the site picks it up; leave it out and the page falls back to the built-in styling. The page is plain, semantic HTML — a `#TOC` nav, one `section` per part and chapter, `.part-label` / `.chapter-label` spans — so a stylesheet has clean hooks to target.

This manual does exactly that: `docs/build.sh` copies its `book4matter-web.css` into `out/book_style.css`, giving the site its own look without disturbing the templates that the PDF, EPUB, and HTML outputs share.

15

IMPORTING FROM WORD

Many manuscripts begin life in Microsoft Word. The `import` command converts a `.docx` into the Markdown this pipeline expects, so you can start from an existing draft instead of retyping it.

```
./run.sh import manuscript.docx mybook/
```

WHAT IT DOES

Pandoc converts the document to Markdown and extracts its images, and book4matter then splits the result into chapter files:

- **One file per Heading 1.** Each `Heading 1` starts a new chapter file. The split is fence-aware, so a `#` inside a code block is never mistaken for a heading. Anything before the first Heading 1 is written to `000-frontmatter.md` for review — usually you trim it, since the template makes its own title page and contents.
- **Numbered in steps of ten.** Files are named `010-slug.md`, `020-slug.md`, and so on, leaving gaps so you can slot in a part divider or an appendix later without renumbering the rest.
- **Images extracted to `media/`.** Pictures are pulled into a `media/` folder and referenced as `../media/...`, the same convention a hand-written book uses. An image Word had embedded in a heading is lifted out to a figure just beneath it.

- **Punctuation kept verbatim.** The author's real em dashes, curly quotes, and ellipses are preserved rather than rewritten, so the Markdown is clean to edit. Tracked changes are accepted and comments are dropped.

FLAGS

- `--no-split` — write the whole document as a single Markdown file instead of splitting on Heading 1.
- `--force` — overwrite a `chapters/` folder that already holds files. By default the importer refuses, so existing work is never clobbered by accident.

AFTER IMPORTING

Expect to tidy up by hand. Import carries the words, the structure, and the images across; you then map the headings onto book4matter's parts and chapters (Word's single level of "Heading 1" cannot express the part/chapter distinction), check the image sizes against the ~300 DPI print target, and trim the front matter. From there it is an ordinary book4matter project.

16

LIMITATIONS

The wonderful thing about standards is that there are so many to choose from.

OUTPUT FORMATS

HTML

HTML is wonderful. The HTML version of this book should work everywhere.

PDF

PDFs are likewise very reliable.

EPUB

EPUB has a lot more issues. Every publisher has slightly different restrictions on things like font encryption, image sizes and more. Book4matter produces clean books that pass the standard epubcheck. Whether it will work with your EPUB publisher is something you will have to check. We value feedback, and strive to make Book4matter more useful. Let us know if you find sharp edges.

PRINT

The *print* option is designed for printing via Amazon KDP, and has been tested there. We have not tested with IngramSpark, Lulu, etc.

Many modern non-print-on-demand printers require PDF/X files, which our tooling doesn't support. They also may want crop marks, bleed marks, color bars, registration marks, etc. You'll probably (currently!) want another tool for going to press. Again, let us know if you find specific things we can improve to make your life easier.

INPUT FORMATS

While writing in Markdown is much simpler than writing in Word, there are some challenges. If you are using human editors, the editing industry largely runs on "Microsoft Word, with track changes". This makes it very easy to have an editor make a lot of changes, and you can easily accept them individually or as a whole.

This actually works just fine with Markdown, as you can use version control and merge. The issue is that the number of book editors who are familiar with Git is **much** smaller than the number who know Word.

AFTERWORD

ORIGINS

Book4matter began due to frustration with Microsoft Word. I wanted to not have to worry about formatting while I was writing, and not worry about how that format would get translated when formatting as a PDF for printing.

Another issue was that writing in Word meant version control really wasn't an option.

Book4matter allows you to write once, in plain text, and publish to multiple formats.

GOALS

Book4matter makes writing my books easier. I hope it will do the same for you. Let us know, via GitHub, about any issues you run into. As with many one-person projects, this project does what I need it to do. Suggestion for how to make it work for you, too, are welcome.

DEPENDENCIES

This project builds on other projects, most notably

- Pandoc (https://pandoc.org), which reads more document formats than anyone reasonably should, and

- Typst (https://typst.app), which makes fine typesetting programmable without a LaTeX-shaped learning cliff.
- We also use epubcheck to validate the output of EPUB generation.