Preface

Faux Press is a book renderer, but it is better to think of it as a document compiler. It takes source files, project metadata, design choices, typography policies, page geometry, and renderer contracts, then emits a PDF that should be readable, inspectable, and defensible.

The first job is not to learn every internal crate. The first job is to make a small book that renders cleanly, then understand where each decision came from. Once that loop is clear, bigger projects become ordinary.

This book is written for the moment when someone says:

> I have Markdown. I want a book-shaped PDF. I want to control the design. I want to know what to do when the output looks wrong.

That is the right starting point. You need a source tree, a manifest, a design reference, a render command, and a validation habit. Everything else grows from those five things.

Part I: The Smallest Working Book

What You Need

The minimum working setup has five parts.

1. A built `faux-press` binary.
2. A book project with `book.toml`.
3. A `src/SUMMARY.md` file that defines reading order.
4. Markdown files for the actual book.
5. A design reference, either built in or external.

That is enough to render a PDF. It is not enough to work comfortably. For real work, add three more things: a stable output folder, a trace file for debugging, and a short visual checklist.

The important habit is to keep source, configuration, and output separate. Source says what the book means. Configuration says how it should be treated. Output shows what survived.

```
book project
  source      → Markdown content
  manifest    → book.toml
  design      → faux-design.yaml or builtin design
  command     → faux-press book render
  artifact    → PDF + optional trace
```

The renderer can do a lot, but a first project should stay plain. Start with chapters, paragraphs, lists, blockquotes, code blocks, and a few small tables. Add figures, citations, custom components, and heavy design overrides only after the basic book loop is stable.

The Mental Model

Faux Press does not simply paste Markdown onto pages. It resolves a document through a chain of questions:

- Source: what did the author provide?
- Structure: what is a chapter, section, paragraph, quote, table, or note?
- Design: which tokens and policies apply?
- Layout: what fits on each page?
- Paint: what exact marks are emitted?
- Trace: why did the system choose that result?

When a page looks wrong, do not guess first. Ask which layer made the decision.

Project Shape

A multi-file Faux Press book follows the same broad shape as an mdBook project:

```
my-book/
  book.toml
  faux-design.yaml
  src/
    SUMMARY.md
    preface.md
    chapters/
      01-start.md
      02-build.md
      03-ship.md
```

The manifest names the book and points Faux Press at the design:

```
[book]
title = "My Book"
authors = ["Author Name"]
language = "en"
src = "src"

[output.faux]
design = "file:faux-design.yaml"
target = "print-a4"
```

`src/SUMMARY.md` is the reading order. In a book-shaped project, this is more than a table of contents. It is also how the renderer learns which Markdown files are book units.

```
# Summary

[Preface](preface.md)

# Part I

- [The First Chapter](chapters/01-start.md)
- [The Second Chapter](chapters/02-build.md)
```

The high-level sections in `SUMMARY.md` can become part-like boundaries. Linked chapter files can become chapter-like units. That matters because chapters usually begin on new pages, while lower headings usually flow on the same page with semantic spacing.

Single File Projects

For a single Markdown file, there is no `SUMMARY.md` to provide boundaries. In that case, the chapter boundary must come from headings or an explicit CLI option:

```
faux-press render manuscript.md \
  --design builtin:book-novel \
  --target print-a4 \
  --chapter-level h1 \
  --output out/manuscript.pdf
```

That keeps the boundary rule explicit. A renderer should not silently guess that every heading is a chapter.

Your First Render

From the repository root, render a book project with:

```
target/debug/faux-press book render
books/getting-started-with-faux-press \
  --target print-a4 \
  --mode print-final \
  --trace \
  --output /tmp/getting-started.pdf
```

The key flags are:

- `book render` uses a multi-file book project.
- `--target print-a4` chooses page geometry.
- `--mode print-final` prefers final-quality layout.
- `--trace` emits the settings and pipeline trace.
- `--output` puts the PDF somewhere deliberate.

For quick previews, use `--mode preview-fast`. For final inspection, use `--mode print-final`. For hard layout debugging, use `--mode exhaustive-debug` when available and when you are willing to pay the time cost.

After rendering, open the PDF and inspect the first ten pages. Do not start with metrics. Start with the artifact:

1. Do chapters begin where expected?
2. Are headings close to the paragraphs they introduce?
3. Are blockquotes visibly separate from surrounding prose?
4. Do code blocks and tables stay inside the allowed page area?
5. Do footnotes or evidence notes appear in the right form?

If the visual answer is wrong, the trace should help explain why.

Where to Put Outputs

For local inspection, use a stable output folder. In this workspace we often use `~/Drop` so rendered PDFs are easy to inspect from other devices.

```
target/debug/faux-press book render
books/getting-started-with-faux-press \
  --target print-a4 \
  --mode print-final \
  --trace \
  --output ~/Drop/getting-started-with-faux-press.pdf
```

Part II: Making It Yours

Designs, Presets, and Taste

A preset is a convenient starting point. A design file is where a project becomes intentional.

```
schema: faux.design/v1
id: my-book-design
extends:
  - builtin:book-novel

intent:
  kind: longform-book
  target: print-a4
  theme: warm-literary
  policyPack: book-print
  componentKit: literary-minimal
  quality: publish

tokens:
  page.margin.left: 90pt
  page.margin.right: 78pt
  spacing.section.clearance: 1.4em
  spacing.quote.before: 1.3em
  spacing.quote.after: 1.3em
```

The useful separation is this:

- `kind` controls the document shape: book, report, essay, manual.
- `target` controls page geometry: A4, trade paperback, mobile.
- `theme` controls visual taste: warm, crisp, editorial, technical.
- `policyPack` controls behavior: chapter starts, paragraph style, keeps, notes.
- `componentKit` controls component variants: quotes, tables, code blocks.
- `quality` controls render intent: draft, proof, publish, debug.

This avoids a bad coupling where a preset secretly controls everything. A novel can use one theme, a technical manual can use another, and a project can override margins without rewriting the renderer.

Paragraph Rhythm

Book prose usually uses first-line indentation and no paragraph gap:

```
policies:
  paragraphs.style: indented-prose
  paragraphs.paragraph_gap: 0pt
```

Web-like documents often use block paragraphs with visible gaps and no first-line indent:

```
policies:
  paragraphs.style: block
  paragraphs.paragraph_gap: 0.75em
```

Do not mix both casually. An indent and a paragraph gap both signal separation. Using both is possible, but it should be a deliberate design choice, not an accident.

Markdown That Typesets Well

Good source gives the renderer clear semantic signals. The goal is not fancy Markdown. The goal is unambiguous structure.

Use headings for structure:

```
# Chapter Title

## Section Title

Body text.
```

Use blockquotes for quoted or callout material:

```
> A trace should tell you what happened, why it happened, and
which rule mattered.
```

Use fenced code blocks for code:

````
```rust
fn main() {
    println!("render the book");
}
```
````

Use tables when the comparison is genuinely tabular:

```
| Need | File |
| --- | --- |
| Reading order | src/SUMMARY.md |
| Project metadata | book.toml |
| Design overrides | faux-design.yaml |
```

Avoid Ambiguous Styling

Markdown is easy to abuse as visual markup. Avoid using bold text as a fake heading. Avoid using blockquotes as indentation. Avoid using code blocks as boxes for ordinary prose. Those choices hide intent from the renderer.

When the renderer knows the role, it can apply the right policy. A heading can keep with the paragraph that follows. A quote can receive quote spacing. A code block can preserve whitespace and use syntax highlighting. A table can choose column widths.

Inline Code

Inline code should be short:

```
Run `faux-press book render` from the project root.
```

Long inline code is harder. It may need wrapping, x-height harmony, and careful line breaking. If the text is a command or a program fragment longer than a phrase, make it a code block.

Validate the Artifact

A finished PDF is not correct because a command exited successfully. It is correct when it preserves the promises of the source, the design, and the output target.

Start with the basic checks:

1. The PDF opens.
2. Page count is plausible.
3. Text is selectable.
4. Chapters and sections appear in order.
5. No visible content overlaps.
6. No content is clipped at page edges.
7. The first few pages look like the intended book.

Then inspect the trace summary. A useful render trace should tell you the resolved settings and late layout warnings.

```
rg -n "glyph page-bound|glyph content-bound" \
  ~/Drop/getting-started-with-faux-press.trace.log
```

Important signals:

- Page-bound glyph overflows should be zero; this means glyphs stayed inside the physical page.
- Glyph-run overlaps should be zero; this means text runs did not collide.
- Page-bound placement overflows should be zero; this means placed regions stayed inside the page.
- Content-bound glyph overflow should be audited when nonzero, because it means text exceeded its safe content box.

Metrics do not replace visual inspection. They narrow the search. The artifact still decides whether the page feels right.

Render and Extract Text

Text extraction catches a different class of error:

```
pdftotext -layout ~/Drop/getting-started-with-faux-press.pdf \
  ~/Drop/getting-started-with-faux-press.txt
```

Use the extracted text to check references, headings, and source order. If the PDF looks right but the text stream is wrong, accessibility and search may still be broken.

Part III: When Something Looks Wrong

Reading the Trace

When a page looks wrong, the trace is the receipt. It should answer three questions:

1. What source object produced this page region?
2. Which settings and policies applied?
3. What layout decision placed it there?

A trace is especially useful when a design decision crosses multiple stages. Block-quote spacing is a good example. The design token may resolve correctly, but the page can still look wrong if a later fallback path ignores the same spacing concept.

That is a connascence problem: two stages silently depend on the same decision, but the decision is not represented as a shared contract.

The fix is not to add a larger magic number. The fix is to make the decision explicit in every stage that consumes it.

Useful Search Patterns

Search for settings:

```
rg -n "tokens.spacing.quote|tokens.page.margin|settings.cross-stage" trace.log
```

Search for flow placement:

```
rg -n "flow-derivation|placement.assignment.fallback|page-break" trace.log
```

Search for visual failures:

```
rg -n "overflow|overlap|clip|fallback|diagnostic" trace.log
```

If a value was configured but not reflected visually, trace both ends. First ask whether the token resolved to the intended value. Then ask whether the stage that places or paints the object used it.

That small discipline prevents most blind debugging.

A Reliable Authoring Loop

The best loop is boring:

1. Edit source.
2. Render to a named PDF.
3. Inspect the first affected pages.
4. Check the trace for warnings.
5. Commit source and config together.

Do not let local experiments become invisible defaults. If a book needs wider margins, put the margin tokens in its design file. If it needs appendix-only evidence, put that policy in `book.toml`. If chapter files should start on fresh pages, encode that as publication role policy.

The source of truth should be boring to find.

```
book.toml          project identity and renderer entrypoint
faux-design.yaml   design tokens and behavior policies
src/SUMMARY.md     book order and chapter boundaries
src/**/*.md        content
trace.log          render receipt
pdf               artifact for human verification
```

What to Version

Version the source and design. Usually do not version every rendered PDF unless the project treats PDFs as release artifacts. If a PDF is important for review, name it clearly and keep it in a review folder such as `~/Drop` or a release directory.

What to Do When It Fails

When a render fails or looks wrong, write down the smallest failing case. A good issue report includes:

1. Source path.
2. Render command.
3. Output PDF path.
4. Trace path.
5. Page number.
6. Expected behavior.
7. Actual behavior.

That turns a visual complaint into a reproducible layout contract.

Appendix A: Command Cheat Sheet

Build the renderer:

```
cargo build -p faux-press --locked
```

Render this book:

```
target/debug/faux-press book render
books/getting-started-with-faux-press \
  --target print-a4 \
  --mode print-final \
  --trace \
  --output ~/Drop/getting-started-with-faux-press.pdf
```

Inspect a book project:

```
target/debug/faux-press book inspect
books/getting-started-with-faux-press
```

Render a single Markdown file:

```
target/debug/faux-press render manuscript.md \
  --design builtin:book-novel \
  --target print-a4 \
  --chapter-level h1 \
  --output ~/Drop/manuscript.pdf
```

Validate a design bundle:

```
target/debug/faux-press design validate file:
books/getting-started-with-faux-press/faux-design.yaml
```

Extract text from a PDF:

```
pdftotext -layout ~/Drop/getting-started-with-faux-press.pdf \
  ~/Drop/getting-started-with-faux-press.txt
```

Generate page images for visual checks:

```
mkdir -p ~/Drop/getting-started-pages
pdftoppm -png -f 1 -l 6 -r 120 \
```

```
  ~/Drop/getting-started-with-faux-press.pdf \
  ~/Drop/getting-started-pages/page
```

Appendix B: Configuration Checklist

Before a book becomes serious, answer these questions.

Project

- Title: the book manifest.
- Author: the book manifest.
- Source directory: the book manifest.
- Reading order: the summary file.

Rendering

- Target page size: the faux output target, or the target command flag.
- Design: the faux output design reference, or the design command flag.
- Quality mode: the render mode command flag.
- Trace output: the trace command flag.

Design

- Margins: left and right page margin tokens.
- Paragraph separation: paragraph style and paragraph gap policies.
- Heading clearance: section clearance spacing.
- Quote spacing: quote before and quote after spacing.
- Code rendering: code highlighting policy.
- Table fit: table sizing and inline leeway policies.

Debugging

- Setting resolution: search for cross-stage settings records.
- Page breaks: search for page-break records.

- Fallback placement: search for placement fallback records.
- Content overflow: search for overflow records.
- Glyph-run overlap: search for glyph-run overlap records.