Table of Contents

1. PDF generation

`prodockit.pdf` builds a standalone PDF from your Zensical site - the kind of downloadable, submittable document professional and academic reports commonly need alongside the website itself. It reads the same `zensical.toml` your site already has, so there's nothing new to learn or configure beyond a couple of optional settings.

1.1 Build your first PDF

From the project root—the directory containing `zensical.toml`—run the `prodockit pdf` command:

```
prodockit pdf
```

The command reads every page in `nav`, keeps that order, and writes `docs/site_documentation.pdf` by default. Open the result and check its cover, contents, headings, page breaks, diagrams, and final page before changing any layout setting.

If the command reports a missing program or native library, install the requirements in the next section and repeat the same command. A project that already completed `prodockit bootstrap --apply` should have them.

1.2 Requirements

The PDF is built via Pandoc and WeasyPrint, so both need to be installed and on your `PATH`:

```
pip install weasyprint
```

then follow Pandoc's own install instructions for your platform (e.g. `brew install pandoc` on macOS).

> ⚠ **WeasyPrint is not a pure-Python package**
>
> `pip install weasyprint` installs the Python half. WeasyPrint draws text through Pango and a few related native libraries - `libgobject-2.0`, `libpango-1.0`, `libpangoft2-1.0`, `libharfbuzz`, `libharfbuzz-subset` and `libfontconfig` - which pip cannot install, because they belong to the operating system.

Platform	
macOS	`brew install pango`
Debian/ Ubuntu	`sudo apt install libpango-1.0-0 libpangoft2-1.0-0 libharfbuzz-subset0`
Windows	`pacman -S mingw-w64-x86_64-pango` under MSYS2, with `C:\msys64\mingw64\bin` on `PATH`

One package covers it on macOS and Windows because glib, HarfBuzz and fontconfig arrive as dependencies of Pango. On Debian, `libharfbuzz-subset0` is a *separate* package from `libharfbuzz0b` and is the one usually missed.

On Apple Silicon macOS, Python's dynamic loader may still not search the Homebrew library directory after Pango is installed. Export the path in the terminal where you run `prodockit pdf`:

```
export DYLD_FALLBACK_LIBRARY_PATH=/opt/homebrew/lib
```

Use `/usr/local/lib` on an Intel Mac. If `cannot load library 'libgobject-2.0-0'` appears after `brew install pango`, check this variable before reinstalling anything.

Missing them looks like a Pandoc problem rather than an install one:

```
Error: pandoc exited with status 43 building 'docs/
site_documentation.pdf' (only pass)

Output from the failing command:
...
OSError: cannot load library 'libgobject-2.0-0'
```

Status 43 is Pandoc's own `PandocPDFError` - Pandoc ran, and the PDF engine it handed off to did not start. The detail beneath the error is WeasyPrint's, and names the library it could not load. `python -c "import weasyprint"` is the quickest way to confirm the stack before building.

Back-of-book indexes additionally need `pymupdf` - `pip install prodockit[index]` (or plain `pip install pymupdf`) - but only if you set `include = true` for `prodockit.index`.

Mermaid diagrams and TeX maths need a little Node tooling on top - see below. Every other feature on this page needs nothing beyond Pandoc/WeasyPrint.

1.2.1 Mermaid diagrams and TeX maths

WeasyPrint has no JS engine, so neither can be rendered the way your live website renders them. Both are pre-rendered to static images before Pandoc sees them - Mermaid via mermaid-cli, maths via a small MathJax script.

Set both up with the `prodockit init-tools` command:

```
prodockit init-tools
```

That writes `tools/mermaid/package.json`, `tools/mathjax/package.json` and `tools/mathjax/tex2svg.js` - the exact layout `prodockit pdf` looks for - then prints the `npm` commands to install them, the `.gitignore` lines you want, and the environment variables a CI run needs. It won't overwrite files you already have unless you pass `--force`, and `--no-mermaid` / `--no-mathjax` skip either half.

Once that's in place, maths is ordinary markdown - write `$...$` for an inline formula and `$$...$$` for a display one, and both are rendered wherever the page is read. The right-angle case, inline: $c = \sqrt{a^2 + b^2}$, and as a display formula:

$$c = \sqrt{a^2 + b^2}$$

Those are live rather than illustrative, and are the only maths in these docs. They earn their place: without a real formula somewhere, this project's own rendering checks cannot tell a working MathJax toolchain from a missing one - `assert_no_unrendered_tex` passes trivially on a document with no maths to leave unrendered. Read them in the PDF and they are static SVG; read them on the website and MathJax typeset them in your browser.

Deliberately shown *rendered* rather than as a fenced markdown sample: a code block's LaTeX and an unrendered formula are the same characters once both are plain text in a PDF, so quoting the source here would fail this project's own check - the same trap that made the "contains TeX maths" warning fire on prose describing it. `assert_no_unrendered_tex` matches the handful of TeX command sequences listed in `prodockit.testing.checks`, so keep literal LaTeX out of any page a `-m built` run inspects - naming even one of them in this sentence was enough to fail the check.

> ⚠ **The failure here is quiet by default**
>
> If a renderer isn't found, the content is left exactly as it is rather than failing the build - the right default for a project using neither feature. A project that *does* use them would otherwise get a PDF full of raw `flowchart LR ...`

source or literal LaTeX with nothing having gone wrong as far as the build is concerned, so since 0.12.0 `prodockit pdf` prints a warning naming the missing renderer whenever that combination occurs.

In CI, use `PUPPETEER_SKIP_DOWNLOAD`

mermaid-cli drives Chrome through Puppeteer. The older `PUPPETEER_SKIP_CHROMIUM_DOWNLOAD` is the one most people reach for, and puppeteer 25.x - what mermaid-cli 11.x resolves to - ignores it, so a full Chrome build is downloaded on every run before being discarded in favour of `PUPPETEER_EXECUTABLE_PATH`. `init-tools` prints the correct pair.

1.3 Configure the PDF

Everything is read from your project's own `zensical.toml` - nothing is passed on the command line beyond, optionally, which config file to use:

```
prodockit pdf --config-file zensical.toml   # -f for short; this is
the default
```

1.3.1 Building a single file

To build a PDF from just one markdown file - a single chapter, say, rather than the whole site - pass `--markdown-file` (`-m` for short), a path relative to `docs_dir`:

```
prodockit pdf --markdown-file chapter1.md   # -m for short
```

This ignores `nav` entirely and renders only that page. Everything else - fonts, page size, margins, `heading_numbering`, and so on - still comes from `zensical.toml` exactly as it would for a full build. The output defaults to that file's own name with a `.pdf` extension inside `docs_dir` (e.g. `docs/chapter1.pdf`) instead of `site_documentation.pdf`, unless `pdf_output` is set, in which case that always wins.

Most of what the PDF needs, it already gets from settings your site likely has for other reasons: `site_name`, `copyright`, `repo_url`, `docs_dir`, `theme.font.text` / `.code`, `theme.icon.admonition`, and `extra_css` - your site's own stylesheet(s) are passed straight through, so a `@media print` rule (e.g. hiding a website-only "Download PDF" link/button, since WeasyPrint always renders in print mode) applies in the PDF too. The rest lives under `[project.extra]`, all optional:

Setting	Default	What it does
`pdf_output`	`"<docs_dir>/site_documentation.pdf"`	Where the PDF is written.
`pdf_copyright`	falls back to `copyright`	Overrides `copyright` for the PDF's own footer only - see Copyright text.
`pdf_page_size`	`"A4"`	Any WeasyPrint-supported CSS page size (`"Letter"`, ...).
`pdf_margin_top` / `_right` / `_bottom` / `_left`	`"2cm"`, except `_bottom` at `"2.5cm"`	Page margins, as CSS lengths. The bottom is deeper because the running footer sits in it - see Copyright text.
`pdf_double_sided`	`false`	Duplex-printing layout - see Double-sided (duplex) printing.
`pdf_margin_inner` / `_outer`	`"2cm"` each	Spine-side/fore-edge margins, used instead of `pdf_margin_left` / `_right` when `pdf_double_sided` is on.
`pdf_header_footer_font_size` / `_color` / `_divider_color`	`"10pt"` / `"#555555"` / `"#e2e8f0"`	Running header/footer styling.
`heading_numbering`	`true`	Chapter/appendix numbering on headings and captions.
`reference_style`	`"european"`	`"european"` (tight, single-line citation entries) or `"global"` (double-spaced, hanging indent - the common APA/MLA/Chicago style).
`pdf_include_table_of_contents`	`true`	Whether to generate and insert a table of contents.
`pdf_table_of_contents_title`	`"Table of Contents"`	That page's own heading text.

Setting	Default	What it does
`pdf_mmdc_bin`	auto-detected	Path to a mermaid-cli `mmdc` binary, for pre-rendering Mermaid diagrams. Diagrams are left unrendered if none is found - see Mermaid diagrams and TeX maths.
`pdf_tex2svg_script` / `pdf_math_dir`	auto-detected	A local MathJax `tex2svg`-style Node script, for pre-rendering TeX math (WeasyPrint has no JS engine to run MathJax client-side). Formulas are left as literal text if none is found - see Mermaid diagrams and TeX maths.
`pdf_extra_css`	none	A list of `docs_dir`-relative stylesheet paths, same shape as `extra_css` above but meant *only* for the PDF - e.g. a rule that would look wrong on the live website, or one overriding something `extra_css` itself sets (concatenated after it, so it wins the cascade).

A page's own front matter `is_appendix: true` gives it letter-based numbering ("A", "A.1", ...) instead of numeric, matching prodockit.headings' own `appendix_attr` convention. A page's own front matter `recto_title: "Short Title"` overrides its running header text from the *next* page onward - see Double-sided (duplex) printing. Your `nav`'s index page can also use `{WORDCOUNT}` / `{REPOURL}` / `{RELEASE}` / `prodockit` markers - see Cover page markers.

1.3.2 Web-only / PDF-only content

Mark any block or inline element `{.web-only}` (via `attr_list`) for content meant only for the live website - a "Download PDF" link/button is the common case, since linking to the very PDF you're already reading doesn't make sense once it's embedded in that PDF. `{.pdf-only}` is the opposite: content meant only for the PDF, e.g. an automated word count or release tag on a cover page that only makes sense in a standalone document.

```
[Download this page as PDF](chapter1.pdf){.web-only}

Word count: {WORDCOUNT}{.pdf-only}
```

`.web-only` needs no configuration - `prodockit.pdf`'s own generated CSS always hides it, in every build, whether you're using `prodockit pdf` or calling `build_pdf()` directly. `.pdf-only` is the one half prodockit can't provide automatically (its own CSS has no reach into your live website), so add this one line to your project's own website stylesheet:

```
.pdf-only {
  display: none !important;
}
```

(see this project's own `docs/stylesheets/extra.css` for a working example). If your project doesn't yet use `.pdf-only` for anything, there's nothing to add until it does.

1.3.3 Copyright text

`copyright` (a native Zensical setting, not one of prodockit's own - see Building a single file above) feeds both the live website's own footer *and* the PDF's running footer, by default - whatever you set once shows, unchanged, in both places.

`pdf_copyright` (under `[project.extra]`) overrides it for the PDF's own footer only, leaving the website's copyright text completely untouched either way. Unset by default, so an existing project's PDF and website keep matching unless you deliberately add it - useful when you want the PDF to show something the website version wouldn't make sense showing (or vice versa), without having to keep two near-identical strings in sync by hand.

Both `copyright` and `pdf_copyright` accept a real HTML fragment, the same as Zensical's own website-side `copyright` setting already does - a real `<a href="../...">` link renders as a real, clickable link in the PDF too, not flattened to plain text. Use a real `<br>` for a forced line break. The obvious use: crediting the tools your report was built with, on their own second line, without touching the copyright/licence text itself:

```
[project.extra]
pdf_copyright = 'Author: Jane Doe. Licensed under the MIT
License.<br>Made with <a href="https://zensical.org/">Zensical</a>
and <a href="https://buckwem.github.io/prodockit-
extensions/">prodockit</a>.'
```

Links and line breaks remain real PDF content rather than being flattened to plain

text. Contributors changing the footer implementation should read PDF pipeline and API.

The equivalent website-side credit (if your project wants a "Made with Zensical and *X*" line on the live site too) isn't a prodockit setting at all - prodockit has no reach into the website's own Jinja partials. It's a Zensical theme override instead: with `custom_dir` set (see Zensical's own docs), drop your own `overrides/partials/copyright.html` based on the bundled version, adding a second credit line after the existing "Made with Zensical" one.

> **Why the bottom margin is deeper than the others**
>
> The footer is top-aligned in the bottom margin and grows *downward* as it gains lines, so whatever the margin does not use is the space left before the paper edge. A two-line footer at a 2cm bottom margin ends about 6.1mm from the edge - inside the 5-6.4mm many consumer and office printers cannot print at all, so the second line risks being cropped even though the PDF itself is correct.
>
> `pdf_margin_bottom` therefore defaults to `2.5cm`, which leaves about 11.1mm. A footer of three or more lines needs more again: set `pdf_margin_bottom` explicitly and check the result on paper, not just on screen.

1.3.4 Cover page markers

Drop any of these literal strings into your `nav`'s index page - typically a cover page, e.g. wrapped in `{.pdf-only}` as in the example above - and `prodockit pdf` substitutes a real value once that page's HTML exists, no configuration needed:

Marker	Becomes
`{WORDCOUNT}`	The site-wide word count (the same value a `54,092` website macro variable would show), so a submission's PDF and its live website page never disagree.
`{REPOURL}`	The git-detected repo URL (the same value `https://github.com/buckwem/prodockit-extensions` gives a website macro).
`{RELEASE}`	The latest published GitHub/GitLab release tag (e.g. `v1.2.0`). The *whole line* containing this marker is dropped instead if there isn't one - most projects never publish a release at all, so nothing shows a bare `"Release: "` label by default.

Marker	Becomes
`prodockit`	Your project's own `site_name`, substituted literally - `prodockit pdf` never evaluates Jinja, so the exact same `prodockit` text a website macro variable uses works here too, one line of markdown for both outputs.

Skipped entirely for a `--markdown-file`-scoped build, or if your `nav` has only one page - there's no separate "cover" vs "content" to compute a word count from either way.

1.3.5 Landscape pages

Anything too wide for a portrait page - a reference table, a diagram, a chart - can be given landscape page(s) of its own instead: wrap it (and its own caption) in `<div class="landscape-page" markdown="1">`, using `md_in_html` (the `markdown="1"` is required - without it, the content inside is left as literal, unconverted text):

```
<div class="landscape-page" markdown="1">

**A wide reference table**

| ID {: width="15%" } | Description {: width="70%" } | Due {:
width="15%" } |
|---|---|---|
| 1 | ... | Q1 |

</div>
```

It is not limited to tables. A Mermaid diagram, an image, a wide code block - whatever is in the block gets the page:

```
<div class="landscape-page" markdown="1">

![Architecture overview](assets/images/architecture.png)

</div>
```

The content prints on its own landscape-sized page(s) - the same configured page size, width and height swapped. A page break is always forced immediately before and after the block, so it never shares a page with anything else.

Content longer than one page simply carries on. A table spanning several landscape pages repeats its header row on every one of them, exactly as it would on a portrait page - measured directly: a 90-row table produced five landscape

pages, each carrying the header (see prodockit.tables for the `width` syntax above, which works the same way here).

A document mixing portrait and landscape pages prints without any special handling - a PDF reader rotates each page to fit the paper on its own.

This is PDF-only - the same wrapped content renders completely normally on the live website, the same way `.web-only` content elsewhere in this project only ever affects one of the two outputs.

1.3.6 Double-sided (duplex) printing

Set `pdf_double_sided = true` under `[project.extra]` for a document meant to be printed and bound on both sides - a book or handbook, rather than a web-printed report. Left-hand (verso) and right-hand (recto) pages mirror their header/footer content and page margins, and every numbered heading starts its own recto page:

```
[project.extra]
pdf_double_sided = true
pdf_margin_inner = "3cm"   # spine side - wider, to leave room for
binding
pdf_margin_outer = "1.5cm" # fore-edge (outer) side
```

`pdf_margin_inner` / `pdf_margin_outer` replace `pdf_margin_left` / `_right` once `pdf_double_sided` is on - the "inner" (spine) side is the left margin on a recto page but the right margin on a verso page, and vice versa for "outer" (fore-edge), so a single pair of settings covers both without you having to think about which physical side is which for any given page. `pdf_margin_top` / `_bottom` are unaffected either way.

Every corner of the running header/footer mirrors between recto and verso, keeping the chapter title and page number on the outer, fore-edge corner and the site name/copyright on the inner, spine-side corner, whichever physical side that happens to be for a given page - confirmed directly, by rendering a real double-sided document and inspecting facing pages, that this is how it actually looks.

Every numbered heading (chapter start) also always starts on its own recto page - a blank page is inserted automatically if the previous chapter ended on an odd page, exactly like the blank pages you'd expect at the start of each chapter in a real printed book. This needs no configuration; it's part of what `pdf_double_sided` turns on.

A page's own front matter `recto_title: "Short Title"` overrides that page's own running header text with a shorter title, from the *next* page onward (the heading's own page still shows its full title) - handy when a chapter's real title is too long to comfortably fit the running header:

```
---
recto_title: "Ch. 1"
---

# Chapter One: A Rather Long Title That Wouldn't Fit In A Running
Header
```

This setting is meaningful whether or not `pdf_double_sided` is on - the running chapter title appears in the header either way, just in a different corner.

1.3.7 Bundling source into a PDF

The `prodockit source-bundle` command builds a second PDF - your Markdown content and `zensical.toml`, one file per page - for a submission that needs the underlying source alongside the rendered document:

```
prodockit source-bundle
```

This is separate from `prodockit pdf` because the two files serve different purposes: one is the rendered document and the other is a record of its source. Run each command only when you need that output.

Writes `docs_dir/source_bundle.pdf` by default, so Zensical serves it with no separate copy step. Override with `pdf_source_bundle_output` under `[project.extra]`:

```
[project.extra]
pdf_source_bundle_output = "dist/source.pdf"
```

The running header's report name is your `site_name`; the page size is `pdf_page_size` - the same setting `prodockit pdf` reads, so both PDFs a project publishes share one physical page size rather than needing it set twice.

Every file is rendered in 8pt Courier with wrapped lines (a genuinely long line wraps rather than running off the page or getting cut off), starting on its own page, with a running header (that page's own file path on the right) and a "Page N of M" footer.

Which files are included: every `.md` file under `docs_dir` (recursively) plus `zensical.toml` itself - your documentation's own source, not the project's tooling around it. A file that isn't valid UTF-8 text is silently skipped rather than failing the build, though in practice that never applies here (Markdown and TOML are always text).

> **Need to bundle more than the document source?**
>
> The command deliberately includes only Markdown pages and `zensical.toml`. Contributors building a custom bundle can use the Python API described in PDF pipeline and API.

1.3.8 Table of contents and bookmark outline

A PDF built by `prodockit pdf` has two separate tables of contents, built by two different tools:

- The **Table of Contents page** itself (`pdf_include_table_of_contents` / `pdf_table_of_contents_title` above) - generated by Pandoc from every heading it sees, via `pandoc.structure.table_of_contents()`.
- The **bookmark outline** - the navigation pane a PDF reader shows down the side, e.g. Adobe Reader's or a browser's own PDF viewer's sidebar. This is built separately by WeasyPrint, which bookmarks every `h1`-`h6` straight from its own UA stylesheet.

prodockit.headings's `unlisted` class (Pandoc's own, not a prodockit invention) keeps a heading off the Table of Contents page. It has no effect on the bookmark outline - an `.unlisted` heading still becomes an outline node, and because outline nesting follows heading level, every later heading of lower level nests underneath it instead of under its real chapter. Add `unbookmarked` too (prodockit's own generated CSS gives `h1.unbookmarked`-`h6.unbookmarked` `bookmark-level: none`) to remove a heading from the outline as well:

```
# Illustrative Example {: .unnumbered .unlisted .unbookmarked }
```

Nothing `prodockit pdf` generates itself carries `unbookmarked` - the back-of-book index's own A/B/C letter headings, the Table of Contents title, and every cover-page heading are all `unnumbered unlisted` (so a reader can still navigate to them) but deliberately *not* `unbookmarked`, since they belong in the outline. Reach for `unbookmarked` yourself only for a heading that shouldn't be there at all - e.g. an illustrative heading in documentation that demonstrates real rendered output rather than a code sample (see this project's own docs/extensions/refs.md).

1.3.9 Back-of-book index

A traditional, two-column back-of-book index, generated from every `\index{Term}` marker when `include = true` in the `prodockit.index` extension's settings. It is PDF-only; there is no equivalent on the live website. Marking terms, turning the setting on, and what the generated page itself looks like are all covered together in Index (PDF only), since (unlike every other feature on this page) marking and generation are two different extensions - see that page

for the full syntax and worked examples. Contributors scripting a custom build can read about the two-pass implementation in PDF pipeline and API.

Contributors calling the Python API or changing the HTML, Lua, CSS, Mermaid, source-bundle, or index stages should use PDF pipeline and API.

1.4 Limitations and workarounds

`prodockit.pdf` pipes your site's own rendered HTML through Pandoc and WeasyPrint to produce the PDF - two tools with their own reader/writer quirks and no JS engine, quite different from a browser rendering your live website. See Known limitations for the confirmed limitations this shapes in `prodockit.pdf.html` / `.lua` / `.css`, and the workaround each one gets.

For supported tool versions, platforms, and the pre-1.0 stability boundary, see Support and compatibility.

Index