Table of Contents

1. Headings

`prodockit.headings` gives every heading in a document an `id` and a hierarchical section number, and records both - along with the heading's text and level - in a shared registry other prodockit extensions build on (e.g. prodockit.refs, which resolves `\ref{id}` by looking an id up in exactly this registry). You can also enable it on its own if you just want ids/numbers on your headings without cross-references.

1.1 Quick start

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.headings"]
```

and every heading gets a hierarchical number automatically - an `h1` is a top-level counter ("1", "2", ...), an `h2` nests under the nearest preceding `h1` ("1.1", "1.2", ...), and so on down through `h6`:

Markdown

```
# Introduction

## Background

## Scope

# Method
```

Result

Heading	id	number
Introduction	`introduction`	`1`
Background	`background`	`1.1`
Scope	`scope`	`1.2`

Heading	id	number
Method	`method`	2

`prodockit.headings` doesn't render the number into the heading text itself - only into the registry above, which prodockit.refs then renders inline wherever you write `\ref{id}`. Numbers are recomputed from scratch on every conversion, so reordering headings always produces correct numbers on the next build - there's no stored/stale numbering state.

1.1.1 Appendices

Flag a page's front matter with `is_appendix: true` to give it letter-based numbering instead of the normal numeric sequence - `"A"`, `"A.1"`, `"A.1.1"` - once you've enabled continuous numbering (see Reference below). An appendix page doesn't consume a number from the numeric sequence at all, so pages after it aren't left with a gap. Letters are assigned sequentially in nav order - the first `is_appendix` page becomes `"A"`, the second `"B"`, and so on, independent of how many numbered pages come before them.

For example, with this nav order:

```
nav = [
  {"Install tooling" = "installtooling.md"},
  {"Glossary" = "glossary.md"},
  {"References" = "references.md"},
]
```

and `glossary.md` flagged as an appendix:

```
<!-- glossary.md -->
---
is_appendix: true
---

# Glossary

## Terms {: #terms }
```

a prodockit.refs reference to `Terms` from another page:

```
<!-- references.md -->
# References
```

```
See \ref{terms} for defined terms.
```

renders to (a link to `glossary.md#terms`, shown here as a code block since `glossary.md` isn't a real page on *this* site):

```
<p>See <a class="prodockit-ref" href="glossary.md#terms">A.1</
a> for defined terms.</p>
```

`Glossary`'s own `h1` becomes `"A"` (the first appendix page in nav) and its `Terms` subheading becomes `"A.1"` - and `References`, the page after it, still gets the next plain number in the numeric sequence (`"2"`, not `"3"`), exactly as if the appendix page had never consumed one. Only meaningful under Zensical; ignored otherwise.

1.1.2 Unnumbered headings

A heading with an `unnumbered` class - e.g. a cover page or title slide - still gets an id, but is skipped when computing section numbers, so it doesn't consume a counter position:

```
# Cover Page {: .unnumbered }

# Introduction
```

`Introduction` above is still numbered `1`, as if `Cover Page` weren't there at all.

1.2 Reference

1.2.1 Continuous numbering across pages (Zensical)

By default, numbering is per-document: every page's own `h1` starts back at 1, regardless of what came before it in a multi-page build. Set `numbering="continuous"` to make `h1` numbering carry on from wherever the previous nav page left off instead:

```
[project.markdown_extensions."prodockit.headings"]
numbering = "continuous"
```

e.g. if page one ends with `h1` number `"3"`, page two's first `h1` becomes `"4"`, not `"1"` again. This is what makes a `\ref{id}` link to a heading on a *different* page (see prodockit.refs) show the same number that's actually displayed on that page.

Once enabled, a page whose front matter sets `is_appendix: true` is numbered with a letter instead - see Appendices above for a worked example.

1.2.2 Ids

An id comes from one of, in order of precedence:

1. An explicit id set via `attr_list`, e.g. `# Introduction {: #custom-id }`.
2. Python-Markdown's own `toc` extension, which `prodockit.headings` enables automatically (with its defaults) if you haven't already enabled it yourself - so if you *have* configured `toc` (e.g. with `permalink: true`), that configuration is left untouched and reused.
3. A minimal built-in slugify fallback, used only if `toc` is somehow not registered at all (this should not normally happen, since `prodockit.headings` enables it).

1.2.3 Options

Option	Type	Default	Description
`source`	`str`	`""`	Identifier for the current document (e.g. its file path). Used to scope this document's entries in the registry, and to safely clear/replace them on a rebuild of the same document.
`registry`	`IdRegistry | None`	a new `IdRegistry()`	Share one registry across multiple documents/ conversions - see below. Passed as a constructor keyword, not a string-based config value (Python-Markdown's config system can't carry arbitrary Python objects safely).
`numbering`	`"per-document" | "continuous"`	`"per-document"`	`"continuous"` makes `h1` numbering carry on across pages in Zensical nav order, instead of restarting at 1 on every page - see above. Only meaningful under Zensical; ignored otherwise.
`appendix_attr`	`str`	`"is_appendix"`	Front matter flag name marking a page for letter-based numbering ("A", "A.1", ...) instead of the normal numeric sequence, when `numbering="continuous"`.

1.2.4 Sharing a registry across a multi-page build

To resolve cross-page references, every page in a build needs to write into - and read from - the *same* `IdRegistry` instance, each scoped by its own, distinct `source`.

Under Zensical, this happens automatically with no configuration - see prodockit.refs for details: `prodockit.headings` detects Zensical's per-page context (each page gets its own fresh `Markdown` instance) and uses it to derive `source` from the page's own path, sharing one registry across the whole build. This auto-detection only activates when you *haven't* set an explicit `registry` or `source` yourself, and has no effect at all outside Zensical.

For any other tool, construct a registry yourself and pass it to every page's extension instance, along with that page's own `source`:

```
import markdown
from prodockit.headings import HeadingsExtension
from prodockit.util import IdRegistry

registry = IdRegistry()

for path, text in pages:
    html = markdown.markdown(
        text,
        extensions=[HeadingsExtension(registry=registry,
source=path)],
    )
```

A duplicate id registered from two *different* sources raises `prodockit.util.DuplicateIdError` here - re-converting the *same* source (e.g. a live-reload dev server) is safe and expected; its previous entries are cleared first. (Zensical's automatic sharing above uses the same registry, but logs a warning and keeps the first registration instead of raising - appropriate for a best-effort default rather than a setup you configured deliberately.)

> ⚠ **The 'keeping the first' winner isn't stable across builds**
>
> A heading name shared across two or more pages (e.g. every page having its own "Quick start"/"Options"/"Syntax" section - common in a set of parallel extension/module docs) produces a "collides with ... - keeping the first" warning under Zensical's automatic sharing above - but *which* page's registration actually wins isn't something you can rely on: Zensical doesn't render pages in a guaranteed stable order, confirmed directly by running `zensical build` repeatedly against identical source and observing the

> reported winner change from one run to the next. Anything depending on that id - a `\ref{id}` /a hand- typed anchor link - can silently point at the wrong page's heading depending on which build produced it.
>
> The fix is to give every colliding heading its own explicit, unique id via `attr_list`, rather than leaving it to whichever page happens to register first:
>
> ```
> ## Quick start {: #refs-quick-start }
> ```
>
> A page-prefixed slug (`<page>-<heading>`) is a simple, collision-proof convention - this project's own documentation uses exactly this scheme throughout (every extension page shares several heading names with the others), so its own markdown source is a worked example if you want to see it applied across a whole site.

1.2.5 Looking up the same numbers from your own build tooling

`prodockit.headings.prescan(appendix_attr="is_appendix")` returns `(start_counts, appendix_letters)` - both `dict[str, ...]` keyed by nav-relative page path - the exact same pre-scan `HeadingsExtension` itself uses internally for `numbering="continuous"`. A consuming project's own build tooling can call this directly to stay in sync automatically, rather than re-deriving the same page-order/heading-count logic a second, independent way - e.g. a template macro that emits a presentational CSS counter-reset per page, matching whatever number `prodockit.headings` computes for that page's first heading (see prodockit.zensical_macros). Returns `None` outside a Zensical build.

1.2.6 CSS hooks

`prodockit.headings` doesn't add any class of its own to a heading - only an `id` (see above), the class(es) already on the heading (e.g. `unnumbered`), and whatever the numbers themselves feed into via the registry (typically consumed by prodockit.refs, or by a template's own build tooling via `prescan()` above to drive presentational CSS). There is currently no `prodockit-heading`-style class to hook a stylesheet onto directly.