Table of Contents

1. Refs

`prodockit.refs` adds a `\ref{id}` cross-reference syntax - similar in spirit to LaTeX's `\ref` - that resolves to the *current* section number of the heading with that id. It depends on the id/number registry that prodockit.headings builds; enabling `prodockit.refs` on its own transparently enables `prodockit.headings` too, with matching defaults, so a single document works with no extra configuration.

1.1 Quick start

Enable it in `zensical.toml`:

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

then reference any heading's id with `\ref{id}`:

```
# Introduction {: #intro }

See \ref{intro} for background.

## Background
```

renders to:

2. Introduction

See 1 for background.

2.1 Background

Because the number is looked up fresh on every conversion, it stays correct even if sections are added, removed, or reordered - you never have to manually renumber a cross-reference. Referencing a heading defined on a *different* page (see Multi-page builds below) links to that page directly (e.g. `other.md#intro`, which Zensical rewrites into the correct clean URL) rather than a bare `#intro` fragment, which would only work if the target happened to be on the same page.

2.1.1 Forward references

A reference to a heading defined *later* in the same document resolves correctly:

```
See \ref{background} below.

## Background {: #background }
```

2.1.2 Unresolved references

`\ref{id}` renders the `unresolved` marker (`??` by default) instead of a number when:

- `id` doesn't exist in the registry at all - e.g. a typo, or a reference to a heading in a page that hasn't been converted yet in a multi-page build (the same way an undefined LaTeX `\ref` shows `??` until a later compilation pass).
- `id` exists but belongs to a heading marked `unnumbered` (see prodockit.headings) - it's still a valid link target in this case, just without a number to show.

```
# Cover Page {: .unnumbered }

See \ref{cover-page}.
```

renders `\ref{cover-page}` as `??`, linked to `#cover-page`.

2.2 Reference

2.2.1 Syntax

```
\ref{<id>}
```

`<id>` is the target heading's id - either one you set explicitly via `attr_list` (`# Introduction {: #intro }`), or the one `toc` derived automatically from the heading text (see prodockit.headings for the exact precedence).

`\ref{...}` is recognised the same way Python-Markdown's own inline syntax is - meaning it's protected inside inline code spans and fenced code blocks, so it's safe to show as a literal example:

````
Type `\ref{intro}` to reference a section.

```

\ref{intro}
```
````

Neither of the two shown above is resolved; both render the literal text.

2.2.2 Options

Option	Type	Default	Description
`unresolved`	`str`	`"??"`	Text rendered when `id` doesn't resolve to a numbered heading.
`source`	`str`	`""`, auto-detected under Zensical	Identifier for the current document (e.g. its path) - used only to decide whether a resolved target is on this same page (bare `#id`) or a different one (a real link to it). Doesn't affect resolution itself.
`registry`	`IdRegistry | None`	discovered from a sibling `prodockit.headings`, or a new one	Share one registry across multiple documents - see below. Passed as a constructor keyword, not a string-based config value.

2.2.3 Multi-page builds

Under Zensical: automatic

Under Zensical, cross-page references work with no extra configuration - just enable both extensions in `zensical.toml` as usual:

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

Zensical builds each page with its own, fresh `Markdown` instance, so `prodockit.headings` detects this (via Zensical's own per-page context) and transparently shares one registry across every page of the build, keyed by each page's own path - no explicit `registry` / `source` needed.

Referencing a heading on a page built later works too, in a single `zensical build` pass: `prodockit.headings` pre-scans every page in the current build's nav for its headings' ids and section numbers before any page has actually been converted, the same way prodockit.citations pre-scans for citation definitions. Pages aren't necessarily built in nav order - or even all within one shared Python process - so without this a cross-page `\ref{id}` could resolve on one build and fall back to `unresolved` (`??`) on the next, from the same unchanged source (see prodockit-extensions#54). A page that *is* converted in the same process supersedes its own pre-scanned entries with the real ones from the parsed document, so the pre-scan only ever fills a gap.

Two pages that happen to share an identically-titled heading (e.g. both have their own "Overview" section) don't fail the build: the *first* one built keeps that id, and the collision is logged as a warning rather than raised as an error - give one of them an explicit id via `attr_list` (`## Overview {: #api-overview }`) to disambiguate and make both referenceable.

Under other tools: manual

Outside Zensical, give `prodockit.headings` and `prodockit.refs` the *same* `IdRegistry` on every page yourself, converting pages in the order cross-references should become resolvable in:

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

registry = IdRegistry()

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

```
        ],
    )
```

Give `RefsExtension` the same `source=path` as `HeadingsExtension` - without it, every resolved link is treated as cross-page (harmless, just not the minimal same-page form for a reference that happens to target its own page).

Here, a genuine id collision between two different `source`s *does* raise `prodockit.util.DuplicateIdError` rather than warning - a deliberately shared registry means you're expected to notice and fix a collision, unlike the best-effort automatic Zensical case above.

2.2.4 CSS hooks

`prodockit.refs` always sets a class on the `\ref{id}` link it renders - resolved or not - so a stylesheet has a stable hook either way:

State	Class
Resolved	`prodockit-ref`
Unresolved	`prodockit-ref prodockit-ref-unresolved`

An unresolved reference (see Unresolved references above) still gets a `class` either way; style `prodockit-ref-unresolved` distinctly (e.g. a warning colour) to make a broken cross-reference visually obvious without inspecting the page source. No `data-*` attribute is left in the rendered output - the internal `data-prodockit-ref` placeholder attribute used during resolution is always stripped before the page is rendered.