Table of Contents

1. Acronyms and Glossary

`prodockit.glossary` lets you define a term once - an acronym expansion, a glossary definition, anything with a short name and a longer explanation - and insert it by id from anywhere in a build, instead of hand-typing a link around the term's own text at every use.

1.1 Quick start

Enable it in `zensical.toml`:

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

Define a term's paragraph with an id and its display text via `attr_list`, then insert it from anywhere with `\gls{id}`:

```
This site uses \gls{css} to control appearance.

**CSS** - Cascading Style Sheets.
{: #css .acronym data-term="CSS" }
```

renders to:

This site uses CSS to control appearance.

CSS - Cascading Style Sheets.

`CSS` is linked directly to the term's own page (e.g. `acronyms.md#css`, or `#css` if used from that same page) - Zensical rewrites that into the correct clean URL for the citing page's own location, the same way a hand-typed `[CSS](acronyms.md#css)` link already gets rewritten.

Unlike prodockit.citations's `\cite{id}`, which *generates* new bracketed citation text (`\cite{id}` → `[Skoulikari, 2023]`), `\gls{id}` inserts the term's *own* registered text in place - closer to LaTeX's `glossaries` package (`\gls{key}` expands to the term's own name) than to a citation. One shared registry covers both acronym entries and glossary entries (and anything else you want to define a short term for) - they're the same kind of thing, an id with a short display text, just conventionally organised across two differently-named pages (see Acronyms and Glossary: one registry, two pages below).

1.1.1 Forward references

A `\gls{id}` pointing at a term defined *later* in the same document resolves correctly, the same way prodockit.refs/prodockit.citations do:

```
See \gls{css} above.

**CSS** - Cascading Style Sheets.
{: #css data-term="CSS" }
```

1.1.2 Unresolved references

An id that doesn't resolve to a definition renders the `unresolved` marker (`?` by default), unlinked:

```
\gls{does-not-exist}
```

renders `?`, with no link.

1.2 Acronyms and Glossary: one registry, two pages

A common convention splits acronym expansions (a short form → long form) and glossary entries (a term → its definition) across two separate pages. `prodockit.glossary` doesn't need to know which page is "acronyms" and which is "glossary" - both are just term definitions in the same registry, so a `\gls{id}` resolves the same way regardless of which page defines it:

```
<!-- acronyms.md -->
**CSS** - Cascading Style Sheets.
{: #css .acronym data-term="CSS" }
```

```
<!-- glossary.md -->
**Cascading Style Sheets** - The language used to control
appearance.
{: #css-def .glossary data-term="Cascading Style Sheets" }
```

1.2.1 Cross-links between entries: use a plain link, not `\gls{id}`

Linking an acronym entry to its own glossary counterpart (and vice versa) is a "see also" cross-reference, not a term insertion - the link text needs to say something like "see the glossary", not repeat the term itself. `\gls{id}` always inserts the *term's own registered text*, so it's the wrong tool here: `\gls{css-def}` renders `Cascading Style Sheets`, so `See \gls{css-def} for what this means` would read *"See Cascading Style Sheets for what this means"* - it resolves, but loses the "go look elsewhere" cue the word "glossary" gives the reader.

Use a plain, hand-typed Markdown link instead, exactly as you would for any other page-to-page cross-reference - it's understood natively by both outputs, and

doesn't need `prodockit.refs` / `prodockit.citations` / `prodockit.glossary` at all:

```
<!-- acronyms.md -->
**CSS** - Cascading Style Sheets. See the [glossary]
(glossary.md#css-def) for what this means in practice.
{: #css .acronym data-term="CSS" }
```

```
<!-- glossary.md -->
**Cascading Style Sheets** - The language used to control
appearance. See the [Acronyms](acronyms.md#css) entry for the
expansion.
{: #css-def .glossary data-term="Cascading Style Sheets" }
```

As a rule of thumb: reach for `\gls{id}` when the term's own name belongs at that point in the sentence, and a plain link when the link text needs to say something else entirely - a "see also", a page name, or any other custom wording.

1.3 Reference

1.3.1 Syntax

Like prodockit.citations, defining and inserting are bundled into one extension: a definition is useless without somewhere to use it.

Defining a term

Any block element - typically a paragraph - with both an `id` and a `data-term` attribute becomes usable with `\gls{id}`:

```
**GUI** - Graphical User Interface.
{: #gui .acronym data-term="GUI" }
```

`data-term` is the text inserted at each `\gls{id}` site - it's stripped from the rendered output (it's internal bookkeeping, not meant to be visible), while `id` stays, since references link straight to it.

Using a term

```
\gls{<id>}
```

Unlike `\cite{...}`, `\gls{...}` only ever takes a single id - there's no multi-term/bracketed form, since inserting a term's own text doesn't compose the way a citation list does.

Like prodockit.refs/prodockit.citations, `\gls{...}` is recognised the same way Python-Markdown's own inline syntax is, so it's protected inside inline code spans and fenced code blocks:

````
Type `\gls{css}` to insert a term.

```
\gls{css}
```
````

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

1.3.2 Options

Option	Type	Default	Description
`source`	`str`	`""`	Identifier for the current document (e.g. its file path). Used to scope this document's own term definitions in the registry, and to build a correct link when a `\gls{id}` target lives on a different page.
`unresolved`	`str`	`"?"`	Text rendered for a `\gls{id}` that doesn't resolve to a definition.
`registry`	`GlossaryRegistry | None`	discovered automatically, or a new one	Share one registry across multiple documents - see below. Passed as a constructor keyword, not a string-based config value.

1.3.3 Multi-page builds

Under Zensical: automatic

Under Zensical, referencing a term defined on a *different* page (the common case - Acronyms/Glossary appendix pages separate from the pages that use them) works with no extra configuration, the same way prodockit.citations shares its registry across pages:

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

Using a term before it's defined works too, the same way as `prodockit.citations`: `prodockit.glossary` pre-scans every page in the current Zensical build's nav for term definitions before any page has actually been

converted, so a term used from an early chapter but defined on an Acronyms/Glossary page kept at the end of nav resolves correctly within a single `zensical build` pass.

Two different sources that happen to define the same id don't fail the build: the first one scanned keeps that id, and the collision is logged as a warning rather than raised as an error.

Under other tools: manual

Outside Zensical, share a `GlossaryRegistry` yourself, the same way as prodockit.citations:

```
import markdown
from prodockit.glossary import GlossaryExtension
from prodockit.util import GlossaryRegistry

registry = GlossaryRegistry()

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

A genuine id collision between two different `source`s raises `prodockit.util.DuplicateIdError` here, rather than warning - a deliberately shared registry means you're expected to notice and fix it.

1.3.4 CSS hooks

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

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

An unresolved id's `<a>` has no `href` (see Unresolved references above) - style `prodockit-gls-unresolved` distinctly (e.g. a warning colour) to make a missing term visually obvious without inspecting the page source. No `data-*` attribute is left in the rendered output - the internal `data-prodockit-gls` placeholder attribute used during resolution, and the `data-term` attribute marking a definition, are both always stripped before the page is rendered.