Table of Contents

- 1. Acronyms and glossary
  - 1.1 Enable the extension
  - 1.2 Define and use a term
  - 1.3 Configure glossary terms
    - 1.3.1 Choose the missing-term text
    - 1.3.2 Use a term before its definition
    - 1.3.3 Fix a missing term
    - 1.3.4 Keep acronyms and glossary terms on separate pages
  - 1.4 Reference
    - 1.4.1 Syntax
    - 1.4.2 Zensical settings
    - 1.4.3 Cross-page terms
  - 1.5 Customise with a CSS style sheet
- Index

1. Acronyms and glossary

`prodockit.glossary` lets you define an acronym or term once and reuse it throughout your documentation. Each use links readers to the definition.

Use it for terms that readers may want to look up, such as an acronym or a specialist word.

1.1 Enable the extension

Enable it in `zensical.toml`:

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

1.2 Define and use a term

Write the definition, then add its id and the text that should appear in your sentences on the line below. Insert the linked term with `\gls{id}`:

Markdown

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

**CSS style sheet** - A file containing Cascading Style Sheets
rules.
{: #css-example .glossary data-term="CSS style sheet" }
```

Result

This site uses a CSS style sheet to control appearance.

CSS style sheet - A file containing Cascading Style Sheets rules.

The extension replaces `\gls{css-example}` with **CSS style sheet** and links it to the definition. Select the link to jump to the definition.

1.3 Configure glossary terms

1.3.1 Choose the missing-term text

An unresolved term displays `?` by default. Set `unresolved` if your project uses a different marker:

```
[project.markdown_extensions."prodockit.glossary"]
unresolved = "MISSING"
```

`source` is the only other TOML setting. It identifies the current page, but Zensical detects it automatically; leave it unset in `zensical.toml`.

1.3.2 Use a term before its definition

You can use a term before its definition appears on the page:

Markdown

```
This example uses a \gls{css-forward-example} for its layout.

**CSS style sheet** - A file containing Cascading Style Sheets
rules.
{: #css-forward-example data-term="CSS style sheet" }
```

Result

This example uses a CSS style sheet for its layout.

CSS style sheet - A file containing Cascading Style Sheets rules.

1.3.3 Fix a missing term

If a term id is missing or mistyped, the extension displays `?` instead of a link:

Markdown

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

Result

?

Check that the text inside the braces exactly matches the id on the definition.

1.3.4 Keep acronyms and glossary terms on separate pages

You can keep acronym expansions on one page and longer glossary definitions on another. `\gls{id}` works with a definition on either page:

Markdown

```
←!── 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" }
```

Result

CSS - Cascading Style Sheets.

Cascading Style Sheets - The language used to control appearance.

Link the two entries

Use an ordinary Markdown link when the link text needs to say “glossary” or “acronyms”. The text inside square brackets is what the reader sees:

```
←!── 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" }
```

Use `\gls{id}` to insert the term itself. Use `[link text](page.md#id)` when you want to choose different words for the link.

1.4 Reference

1.4.1 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

Syntax	Purpose
`\gls{<id>}`	Insert one term's registered display text and link it to its definition

Unlike `\citeref{...}`, `\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.

1.4.2 Zensical settings

Setting	Default	What it controls
`unresolved`	`"?"`	Text shown for a term id that cannot be found.
`source`	`""` (detected automatically)	Advanced: identifies the current page when using the extension outside Zensical. Leave it unset in `zensical.toml`.

1.4.3 Cross-page terms

Under Zensical, a term can be used on a different page from its definition, including an Acronyms or Glossary appendix later in navigation. Prodockit reads definitions across the navigation before page conversion, so no author configuration is required.

Two definitions using the same id produce a warning and the first definition is retained. Use a unique id for every term.

For integration with another Markdown renderer, see Extension integration.

1.5 Customise with a CSS style sheet

`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.

Index

P

`prodockit.glossary`, 2
  `source`, 6
  `unresolved`, 6