Table of Contents

1. Citations or References

`prodockit.citations` lets you define a source once and cite it by key from anywhere in a build, instead of hand-typing a bracketed link at every citation site.

> **Looking for an auto-generated bibliography instead?**
>
> `prodockit.citations` is one of two alternatives for citing sources - this one resolves against a hand-authored paragraph you write yourself, once. If you'd rather define sources in a BibTeX/BibLaTeX `.bib` file and have the reference list generated for you, in any citation style, see prodockit.bibliography and its comparison of both approaches.

1.1 Quick start

Enable it in `zensical.toml`:

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

Define a source's paragraph with an id and a short display text via `attr_list`, then cite it from anywhere with `\cite{id}`:

```
Git is a tool used to manage version control.\cite{skou2023}

Skoulikari, A. (2023) *Learning Git: A Hands-On and Visual
Guide to the
Basics of Git*. Sebastopol, CA: O'Reilly Media.
{: #skou2023 .reference data-cite-text="Skoulikari, 2023" }
```

renders to:

Git is a tool used to manage version control.[Skoulikari, 2023]

Skoulikari, A. (2023) *Learning Git: A Hands-On and Visual Guide to the Basics of Git*. Sebastopol, CA: O'Reilly Media.

`[Skoulikari, 2023]` is linked directly to the source's own paragraph (e.g. `references.md#skou2023`, or `#skou2023` if cited 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 `[text](references.md#skou2023)` link already gets rewritten. Unlike hand-typing that link yourself, you never have to work out the relative path to the references page, retype the display text, or fix every citation site if the display text needs to change - it's defined once.

Copyright © 2026 Mark Buckwell and contributors.
Made with Zensical and prodockit.

Multiple comma-separated keys join into one bracket: `\cite{skou2023,chacon2014}` → `[Skoulikari, 2023; Chacon and Straub, 2014]`.

Unlike prodockit.glossary's `\gls{id}`, which inserts a term's own registered text in place, `\cite{id}` *generates* new bracketed citation text around a link - closer to a bibliography citation than a glossary/acronym expansion.

1.1.1 Forward references

A citation to a source defined *later* in the same document resolves correctly:

```
See \cite{skou2023} above.

Skoulikari, A. (2023) *Learning Git*.
{: #skou2023 data-cite-text="Skoulikari, 2023" }
```

1.1.2 Unresolved citations

A key that doesn't resolve to a definition renders the `unresolved` marker (`?` by default) in place of that one entry - the rest of a multi-key citation still resolves normally:

```
\cite{skou2023,does-not-exist}
```

renders `[Skoulikari, 2023; ?]` - the unresolved entry has no link, unlike a resolved one.

1.2 Reference

1.2.1 Syntax

Defining and citing are bundled into one extension, unlike prodockit.headings/prodockit.refs: a definition is useless without somewhere to cite it, so there's no independently useful "just defining" half to split out.

Defining a source

Any block element - typically a paragraph - with both an `id` and a `data-cite-text` attribute becomes a citable source:

```
Chacon, S. and Straub, B. (2014) *Pro Git*. 2nd edn. New York:
Apress.
```

Copyright © 2026 Mark Buckwell and contributors.
Made with Zensical and prodockit.

```
{: #chacon2014 .reference data-cite-text="Chacon and Straub,
2014" }
```

`data-cite-text` is the short text rendered at each citation site - it's stripped from the rendered output (it's internal bookkeeping, not meant to be visible), while `id` stays, since citations link straight to it.

Citing a source

```
\cite{<id>}
\cite{<id1>,<id2>,...}
```

Like prodockit.refs, `\cite{...}` 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 `\cite{skou2023}` to cite a source.

```

\cite{skou2023}
```
````

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

1.2.2 Options

Option	Type	Default	Description
`source`	`str`	`""`	Identifier for the current document (e.g. its file path). Used to scope this document's own citation definitions in the registry.
`unresolved`	`str`	`"?"`	Text rendered for a `\cite{id}` key that doesn't resolve to a definition.
`registry`	`CitationRegistry | 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.

1.2.3 Multi-page builds

Under Zensical: automatic

Copyright © 2026 Mark Buckwell and contributors.
Made with Zensical and prodockit.

Under Zensical, citing a source defined on a *different* page (the common case - a references page separate from the pages that cite it) works with no extra configuration, the same way prodockit.refs shares its registry across pages: `prodockit.citations` detects Zensical's per-page context and shares one registry across the whole build automatically.

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

Citing a source before it's defined works too - the common case, since a references page is usually cited from earlier chapters but kept at the *end* of nav as an appendix. Normally that's a forward reference to a page `zensical build`'s single, one-shot pass hasn't rendered yet (unlike `zensical serve`'s live-reload, which eventually rebuilds every page at least once) - `prodockit.citations` avoids this by pre-scanning every page in the current Zensical build's nav for citation definitions (reading raw file text directly, not waiting for Python-Markdown to parse each one) before any single page has actually been converted, the same way LaTeX needs multiple compilation passes to resolve a `\cite` used before its `\bibitem` - except here it happens automatically, within one `zensical build` invocation.

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

Under other tools: manual

Outside Zensical, share a `CitationRegistry` yourself, the same way as prodockit.headings:

```
import markdown
from prodockit.citations import CitationsExtension
from prodockit.util import CitationRegistry

registry = CitationRegistry()

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

A genuine key 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.

Copyright © 2026 Mark Buckwell and contributors.
Made with Zensical and prodockit.

1.2.4 What this doesn't do (yet)

`prodockit.citations` covers citation-key management - the "define once, cite anywhere" part. It doesn't auto-generate the references page's listing itself from structured bibliographic data (author/year/title/publisher/URL fields) the way a full BibTeX-style tool would - the reference entry's own text (as shown in the examples above) is still hand-authored prose, just like today. See prodockit.bibliography for the alternative that does exactly this, from a `.bib` file, in any citation style - and for the tradeoffs between the two approaches.

1.2.5 CSS hooks

`prodockit.citations` emits three hooks - one on the outer wrapper, one on each individual key's own link:

Element	State	Class
Outer `<span>` wrapping the whole `\cite{...}` citation	always	`prodockit-cite`
Each key's own `<a>`	resolved	`prodockit-cite-resolved`
Each key's own `<a>`	unresolved	`prodockit-cite-unresolved`

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

Copyright © 2026 Mark Buckwell and contributors.
Made with Zensical and prodockit.