

Table of Contents

1. Index (PDF only)	2
1.1 Choose between tags and index terms	2
1.2 Before you start	3
1.3 Enable the extension	3
1.4 Mark a term	3
1.5 Configure the generated index	4
1.5.1 Add sub-entries	4
1.5.2 Add code-styled terms	5
1.5.3 Add linked terms	6
1.6 Reference	7
1.6.1 How index generation works	7
1.7 Customise with a CSS style sheet	8

1. Index (PDF only)

`prodockit.index` marks important terms where they are discussed and uses those markers to build an alphabetised, PDF-only back-of-book index with page numbers. The marked words remain ordinary inline text on the website, where readers can use Zensical's site search instead.

Use this extension for concepts a printed reader may need to find again. Use [Acronyms and glossary](#) when a term needs a separate definition that readers can follow as a link. The actual index page is generated by [prodockit.pdf](#); marking terms alone does not add that page.

1.1 Choose between tags and index terms

[Zensical tags](#) and `prodockit.index` both help readers find related material, but they solve different problems. Tags classify a whole page for website discovery; index markers identify the exact places where a printed reader should look. The differences are summarised in [Table 28.1](#).

Table 1.1. Zensical tags and Prodockit PDF index terms

Capability	Zensical tags	<code>prodockit.index</code>
Granularity	Entire page	Exact occurrence within a page
Main output	Tag linked to relevant website pages	Term linked to rendered PDF page numbers
Hierarchy	Nested tags, where supported	<code>Parent!Child!Grandchild</code> sub-entries
Repeated entries	One page appears under a tag	Occurrences are deduplicated into page numbers and ranges
Primary purpose	Website discovery and categorisation	Traditional printed back-of-book index

Use tags when a reader would benefit from discovering all pages about a broad subject. Continue to use `\index{...}` for precise terms, commands and concepts that should lead to their actual location in the generated PDF. A project can use both: tags improve website navigation and search, while index markers build the PDF index without displaying extra labels on the website.

Zensical 0.0.58 introduced replacements for the Material for MkDocs [tags plugin](#), but some details are still being established:

- Zensical's tags documentation currently describes page tags and search filtering but says that generated tag listings are not yet supported. Confirm the behaviour of the project's pinned Zensical version before relying on a website tags-index page.

- A tags listing maps a subject to pages, not to the exact physical pages produced during PDF layout. It therefore cannot currently replace `prodockit.index` page-number collection or page-range collapsing.
- Prodockit does not currently convert page tags into PDF index entries. Such a bridge could provide optional, coarse page-level entries, but its handling of nested tags, long pages and controlled vocabularies needs designing and testing first.
- It is not yet clear whether a generated tags listing should be included in a combined PDF. Even if included, it would be a subject listing rather than a conventional back-of-book index.

1.2 Before you start

No separate installation command is needed. When an enabled back-of-book index is first used, `pdk pdf` adds PyMuPDF to the project's PDF preparation and records it in the project-local cache. Run `prodockit config` to confirm that index generation is enabled; missing first-use PDF packages are deferred rather than reported as configuration errors.

1.3 Enable the extension

Enable it in `zensical.toml`:

```
[project.markdown_extensions."prodockit.index"]
include = true
```

1.4 Mark a term

Mark a term with `\index{Term}`:

Markdown

A `\index{widget}` is the basic unit of work.

Later, this `\index{widget}` gets combined with a `\index{gadget}`.

Result

A widget is the basic unit of work.

Later, this widget gets combined with a gadget.

Every marked term renders inline exactly as written - `\index{widget}` becomes plain "widget" text, nothing more, on the live website. The marker only has an effect on the *PDF*, and only once `include = true` is set - see [Generating the index](#) below for the generated index page itself.

Marking the same term more than once creates one index entry with all of its page numbers.

1.5 Configure the generated index

Set `include = true` in the extension's `zensical.toml` table for a traditional, two-column back-of-book index - terms grouped under a bold letter heading (A, B, C, ...), each followed by the page number(s) it appears on - appended as its own page(s) at the very end of the document:

```
[project.markdown_extensions."prodockit.index"]
include = true
title = "Index"
```

[Table 28.2](#) explains the two settings used by that configuration.

Table 1.2. Configure the generated index

Setting	Default	What it controls
<code>include</code>	<code>false</code>	Whether <code>prodockit pdf</code> generates the index pages.
<code>title</code>	<code>"Index"</code>	Heading shown on the first generated index page.

You can leave out `title` when "Index" is the heading you want.

The `\index{widget} / \index{gadget}` example above renders to an index page like:

```
Index

G
Gadget, 3

W
Widget, 1, 3
```

with its own page list deduplicated and sorted; consecutive pages collapse into an en-dash range (`67-70`) rather than listing every page individually, and non-consecutive pages/ranges are comma-separated (`64, 175`) - standard back-of-book index convention.

A term is alphabetised (and letter-grouped) ignoring any leading punctuation - `--set-upstream` option (`git push`) and `-u` option (`git branch`) are filed under **S** and **U** respectively (matching where `"set-upstream"/"u"` itself would sort), not lumped into a separate "symbols" section, the same way a technical book's own index treats command-line options.

1.5.1 Add sub-entries

`\index{Parent!Child!Grandchild}` - `!` separates up to three levels in practice, matching

LaTeX `makeidx`'s own long-established `\index{primary!secondary!tertiary}` convention - nests a term under another, the same way a printed book's own index groups related entries together (e.g. "staging area" with "adding"/"modified" indented beneath it) rather than listing every term as one flat alphabetical run:

Markdown

```
Now generate the \index{Git!ssh keys} to use for authentication.
```

Result

Now generate the ssh keys to use for authentication.

Only the *last* segment displays inline (`ssh keys` above) - wherever the term is actually mentioned in your prose - the earlier segments (`Git`) are only ever used to build the generated index's own nesting, and never appear inline themselves. Renders a nested entry like:

```
G
  Git
    ssh keys, 13, 89
```

A parent with no marker of its own anywhere (like `Git` above) still gets its own line - a bare category label with no trailing page list - so its children have somewhere to nest under.

1.5.2 Add code-styled terms

Backticks around the *last* segment mark a command or other code term - it displays inline in a real `<code>` element instead of plain text, and the generated index entry renders the same way:

Markdown

```
Run \index{`git commit`} to save your changes.
```

Result

Run `git commit` to save your changes.

Combine this with [sub-entries](#) by putting the backticks around just the last segment - nesting a code-styled `git commit` entry under a plain `Git` one:

```
\index{Git!`git commit`}
```

Both render an index entry in your document's own monospace font, matching how the term already displays inline:

G

Git

git commit, 13, 89

Showing this syntax as literal example text

Unlike a plain `\index{Term}` (protected by a real code span, the same way this page's own examples above are written), the code-styled pattern has to run *before* Python-Markdown's own backtick handling, so it can recognise its own inner backticks - a side effect is that wrapping the whole call in inline backticks doesn't protect it the way it does for the plain syntax. A fenced code block (as every example on this page already uses) still works, since it's stashed before any inline pattern - this one included - ever runs.

1.5.3 Add linked terms

A term can be a Markdown link. The link remains clickable in the text while the index files it under the visible link text:

Markdown

```
\index{[Git](https://git-scm.com/)} is a version control system.
```

Result

[Git](#) is a version control system.

 attr_list ({target="_blank"}) doesn't combine cleanly

Do not put `{target="_blank"}` after a linked index marker; it attaches to the marker rather than the link. If the link must open a new tab, use a raw inline `<a>` tag:

```
\index{<a href="https://git-scm.com/" target="_blank">Git</a>} is a version control system.
```

1.6 Reference

[Table 28.3](#) shows how each authoring form appears inline and where it is placed in the generated index.

Table 1.3. Reference

Syntax	Inline result	Index structure
<code>\index{Term}</code>	Term	One flat entry
<code>\index{Parent!Child}</code>	Child	Child nested under Parent
<code>\index{Parent!Child!Grandchild}</code>	Grandchild	Up to three levels
<code>\index{`command`}</code>	Code-styled command	Code-styled flat entry
<code>\index{Parent!`command`}</code>	Code-styled command	Code-styled child entry
<code>\index{[Text](url)}</code>	Linked Text	Entry filed as Text

Important constraints:

- `!` separates hierarchy levels; only the final segment appears in the prose.
- Backticks for code styling belong around the final segment.
- Use a fenced code block, not an inline code span, when documenting the code-styled marker literally.
- `attr_list` attributes do not attach cleanly to a Markdown link inside an index marker; use a raw `<a>` element when a linked term needs attributes.

1.6.1 How index generation works

Index generation uses two PDF passes. The first pass determines the page of each marked term; the second adds the completed, deduplicated index. Document authors do not need to

configure those passes. Contributors changing the pipeline should read [Extension integration](#).

1.7 Customise with a CSS style sheet

Every marked term has the class `index`. Target `.index` if you want marked terms to look different on the website. A code-styled term also contains a normal `<code>` element, so it keeps the project's existing code style.

The attributes used to construct the generated PDF index are an internal pipeline contract. Contributors changing them should read [Extension integration](#).