markpublish User Manual

Modern PDF Document Publishing from Markdown

> *Practical guide and reference for configuration, CLI usage, theme design, and document publishing with markpublish.*

Version 2.1.5
Date 2026-09-21
Author Frank Winter

Table of Contents

CHAPTER 1

Introduction

Motivation, core features, and architecture of markpublish.

AT A GLANCE

1.1 Motivation and Objectives

Markdown is the leading format for technical documentation, notes, and project reports. Its strength lies in simplicity and readability in plain text. However, when these texts need to be turned into high-quality, print-ready reports, manuals, or whitepapers, traditional tools quickly reach their limits. Authors often face time-consuming manual post-processing in word processing or desktop publishing software.

markpublish closes this gap: it automatically transforms structured Markdown files via a simple yet powerful configuration into typographically refined, professionally designed PDF documents.

The goal of markpublish is to provide developers, technical writers, and authors with a seamless publishing workflow. Content is written in plain Markdown, while markpublish takes full control over layout, document structure, divider pages, numbering, tables of contents, and typography.

1.2 Core Features

markpublish focuses on document quality, speed, and reliability:

- **Modern Typesetting Quality with Typst:** Powered by the innovative Typst typesetting engine, documents feature outstanding typographical precision, balanced margins, and aesthetic layouts.
- **Blazing Compilation Speed:** Even comprehensive documents with dozens of pages, illustrations, and tables compile in about one to two seconds.
- **Simple Installation and Portability:** As a standard Python package, markpublish runs cross-platform on Windows, macOS, and Linux without complicated dependencies.
- **Declarative Configuration:** A single file (`markpublish.yaml`) controls the entire publishing project.
- **Multi-Tier Document Architecture:** Clear distinction between high-level sections (*Parts*) and content chapters (*Chapters*) with flexible divider page control.
- **Precise Tables of Contents and Numbering:** Fully automated generation of document-wide, part-level, and chapter-level tables of contents, alongside customizable chapter and heading numbering.
- **Comprehensive Markdown Extensions:** Built-in support for GitHub-style callout boxes (admonitions), tables, definition lists, footnotes, task lists, and syntax-highlighted code blocks.
- **Cascading Internationalization (i18n):** Integrated support for multilingual documents and themes (static text labels such as Table of Contents, Chapter, page numbers) through

a four-tier cascade. Furthermore, the command-line interface (CLI) is fully bilingual (English and German) and automatically adapts to your operating system's language.

1.3 Architecture Overview

markpublish operates along a clean processing pipeline:

1. **Configuration and Document Analysis:** markpublish reads `markpublish.yaml`, validates all settings, and constructs the hierarchy of parts and chapters.
2. **Content Processing:** Chapter Markdown files are parsed via an ElementTree AST pipeline and converted directly into clean Typst markup by a native Typst serializer (`TypstSerializer`). Headings and tables of contents are synchronized at the AST level, link targets are resolved, and local image paths are automatically isolated for the build.
3. **Template Assembly:** The selected theme (by default the built-in standard theme) provides layout definitions. Document metadata (packaged in a typed `meta` dictionary), headers, footers, divider pages, and body content are mapped directly into the Typst environment.
4. **PDF Compilation:** The Typst engine compiles the assembled document directly into a finished PDF file in a single, highly efficient run. If an error occurs, the generated Typst source code is saved to `.markpublish/last_failed_build.typ` for immediate debugging.

CHAPTER 2

Installation and Quickstart

> The direct path from installation to your first finished PDF.

AT A GLANCE

This chapter outlines the fastest path from installation to your first rendered PDF document.

2.1 Prerequisites

markpublish requires a standard Python environment:

- **Python version:** 3.10 or newer
- **Operating system:** Windows, macOS, or Linux

2.2 Installation

Install markpublish via Python's package manager `pip`:

```
pip install markpublish
```

After installation, the `markpublish` command (along with the shorthand alias `mpub`) is ready in your terminal. You can verify the installation by checking the version:

```
markpublish --version
```

2.3 Steps to Your First Build

Setting up a new document project takes just a few quick steps:

2.3.1 Initialize a New Project

The `init` command creates a new project directory with a minimal, runnable starter template:

```
markpublish init my-document
```

If the specified directory already exists and is not empty, `init` safely aborts to prevent accidental overwriting of existing files.

This command creates a folder named `my-document/` with the following structure:

```
my-document/
├── markpublish.yaml
└── welcome.md
```

2.3.2 Build Your First PDF

You can build the PDF directly from the parent directory by passing the path to the configuration file (which `init` displays as a recommendation upon completion):

```
markpublish build my-document/markpublish.yaml
```

Alternatively, switch into the newly created folder and trigger the build there:

```
cd my-document
markpublish build
```

markpublish parses `markpublish.yaml`, processes all referenced Markdown files, and compiles the document.

2.3.3 Open the Document

The generated PDF is placed directly inside your project folder:

```
my-document/
└── my_document.pdf
```

(Note: The output filename is automatically derived from the document title configured in `markpublish.yaml` – spaces and hyphens are converted to underscores. Without `--title`, the new project takes the directory name, so `my-document` becomes `my_document.pdf`.)

Open the file in any PDF viewer to inspect the result. The document features the formatted starter chapter complete with header, footer, and typographic styling.

CHAPTER 3

Command-Line Interface (CLI)

> Detailed command overview and complete option reference for all commands.

AT A GLANCE

The command-line interface is the primary way to interact with markpublish. In addition to the main `markpublish` command, the shorthand alias `mpub` is available.

3.1 Command Overview

Command	Summary
`build`	Compiles the document based on the configuration file.
`init`	Creates a new project with a configuration file and starter chapter.
`manual`	Renders the official user manual from the installed package.
`cheatsheet`	Renders the compact two-page quick reference from the package.
`templates`	Lists all available templates and their origin layers.
`export-template`	Exports the default theme into the project for customization.
`labels`	Shows all resolved text labels and their origin layer in the i18n cascade.

3.2 markpublish build

The `build` command runs the publishing pipeline. It loads the configuration, processes all chapter Markdown files, and compiles the final document.

```
markpublish build [CONFIG_FILE] [OPTIONS]
```

3.2.1 Options

Option	Default	Description
`CONFIG_FILE` *(Argument)*	`markpublish.yaml`	Path to the project configuration file.
`--target`, `-t`	`pdf`	Output format (`pdf`).
`--output`, `-o`	*(automatic)*	Path to output file or destination directory. If the path has no file extension, it is treated as a directory.
`--theme`	`None`	Renders with an alternative theme instead of the one specified in the configuration.

Option	Default	Description
`--templates-dir`	`None`	Path to a custom templates directory.

3.2.2 Details

Without arguments, markpublish looks for `markpublish.yaml` in the current working directory – inside a project folder, running `markpublish build` is sufficient. If a custom path is specified, its directory is treated as the project root: all relative paths in the configuration (chapter files, images, templates) are resolved from that directory, not from your current working directory.

If `--output` is omitted, markpublish writes the file into the project folder. The output filename is derived automatically from the document title (e.g., `my_document.pdf`).

`--templates-dir` expects the container directory, not an individual theme folder. Which theme is used is determined either by the `--theme` command-line option or by `theme:` in `markpublish.yaml` (default: `default`). markpublish combines both settings and looks for `<directory>/<theme>/<target>`:

```
my-templates/           <- pointed to by --templates-dir
└── corporate/          <- named by theme: in markpublish.yaml
    └── pdf/            <- target format
        ├── template.typ
        └── i18n.yaml
```

The corresponding command is:

```
markpublish build --templates-dir my-templates
```

A relative path is resolved relative to the project directory. The same directory can also be configured permanently; markpublish checks these locations in order:

1. `--templates-dir` on the command line
2. `templates_dir:` in `markpublish.yaml`
3. The environment variable `MARKPUBLISH_TEMPLATES_DIR`
4. A `templates/` folder next to the configuration file

The directory determined this way is only one of three tiers searched by markpublish; a template with the same name in your user home directory still takes precedence (see chapter *Theme System and Internationalization*).

Examples:

```
# Standard build in current project
markpublish build
```

```
# Build with an alternative theme (overrides markpublish.yaml)
markpublish build --theme custom-theme

# Specify custom configuration file and output directory
markpublish build projects/report.yaml --output dist/
```

3.3 markpublish init

The `init` command scaffolds a new project quickly. It creates the destination folder (if not yet present), a runnable `markpublish.yaml`, and an initial Markdown chapter.

```
markpublish init [DEST_DIR] [OPTIONS]
```

3.3.1 Options

Option	Default	Description
`DEST_DIR` *(Argument)*	`.` *(current directory)*	Target directory in which the project should be initialized.
`--title`, `-t`	*(Destination folder name)*	Document title written into the generated configuration.
`--lang`, `-l`	*(System language)*	Language of the project template (`en`, `de`).

3.3.2 Details

The generated `markpublish.yaml` and starter chapter are copied from built-in templates. You can specify the template language explicitly with `--lang` (e.g., `--lang en`); without this flag, your system language is detected automatically, ensuring the project compiles consistently on any machine.

If the target directory already exists and contains files, `init` aborts with an error message. An existing directory is never overwritten silently; to reinitialize, choose a new directory name or make sure the folder is empty.

Example:

```
markpublish init my-manual --lang en --title "System X User Guide"
```

3.4 markpublish manual / markpublish cheatsheet

markpublish packages its complete documentation directly within the software. Both commands compile documentation on-demand from the installed package sources – the generated document always reflects the exact installed version.

- `markpublish manual`: Renders this comprehensive user manual.
- `markpublish cheatsheet`: Renders a compact two-page reference.

```
markpublish manual [OPTIONS]
markpublish cheatsheet [OPTIONS]
```

3.4.1 Options

Option	Default	Description
`--lang`, `-l`	*(System language)*	Document language (`en`, `de`).
`--target`, `-t`	`pdf`	Output format (`pdf`).
`--output`, `-o`	`.` *(current directory)*	Output destination path.
`--theme`	`None`	Renders with an alternative theme instead of the default theme.
`--templates-dir`	`None`	Path to a custom templates directory.

3.4.2 Details

If no `--lang` option is provided, markpublish detects your operating system language. If no translation exists for that language, markpublish falls back to the document's default language.

Example:

```
# Render the English user manual in the current directory
markpublish manual --lang en

# Render the German cheat sheet
markpublish cheatsheet --lang de
```

3.5 markpublish templates

Lists all templates available across your system and project, clearly indicating precedence when multiple templates share the same name.

```
markpublish templates [OPTIONS]
```

3.5.1 Options

Option	Default	Description
`--target`, `-t`	`None`	Filters the list by output format (`pdf`).
`--templates-dir`	`None`	Additional template search directory.

3.5.2 Details

Outputs a structured table showing the theme name, origin layer (User, Common, or Package), and a status indicator showing which theme would be activated during a build.

3.6 markpublish export-template

Exports a packaged theme into a local directory so it can be customized or used as the foundation for your own corporate design.

```
markpublish export-template [THEME] [DEST_DIR] [OPTIONS]
```

3.6.1 Options

Option	Default	Description
`THEME` *(Argument)*	`default`	Name of the theme to export.
`DEST_DIR` *(Argument)*	`templates`	Destination directory for exported template files.
`--target`, `-t`	`all`	Target format to export (`pdf` or `all`).

3.6.2 Details

During export, the Typst template files (`template.typ`) and the accompanying `i18n.yaml` label file are copied into the target directory. When a theme is located inside the project's `templates/` directory, markpublish automatically picks it up during builds.

Example:

```
# Export the default theme into the local ./templates directory
markpublish export-template default ./templates
```

3.7 markpublish labels

Displays a detailed report of all static text labels (such as "Table of Contents", "Chapter", "Page X of Y") for the project's selected document language, along with their resolution layer in the cascade.

```
markpublish labels [CONFIG_FILE] [OPTIONS]
```

3.7.1 Options

Option	Default	Description
`CONFIG_FILE` *(Argument)*	`markpublish.yaml`	Path to the project configuration file.
`--target`, `-t`	`pdf`	Target format whose label cascade should be inspected.
`--theme`	`None`	Inspects the label cascade for an alternative theme.
`--templates-dir`	`None`	Custom templates directory.
`--overridden`	`False`	Displays only labels that have been overridden by a theme or project layer.

3.7.2 Details

This command is the primary diagnostic tool when developing themes or configuring documents. Before running a build, it verifies that all metadata fields, text labels, and theme function signatures match up completely.

The report is structured into eight columns:

Column	Description
Key	Name of the metadata or text label key (e.g., `title`, `author`, `toc_title`, or custom fields like `department`).
Theme Usage	How the theme references this key: `key` (as a label identifier), `value` (as content value), `key/value` (both), or `-` (unused by theme).
Label	Resolved text string from the i18n cascade for this key. Missing required labels are highlighted in red.
i18n Source	Cascade layer providing the label (`mpub`, `theme`, `target`, `project`). Layers that override defaults are highlighted in green.
Label Fallback	Fallback text defined inside the theme if the label does not exist in any i18n layer.
Value	Content value defined in the document configuration (`document:`).
Value Fallback	Fallback value defined in the theme if the field is omitted in the document.
Status	Diagnostic assessment: `OK`, `unused`, `Missing label`, `Value not set`, or critical errors in red that would prevent compilation.

The identifiers in the **i18n Source** column represent:

Value	Source File
`mpub`	Built-in package default, `markpublish/i18n.yaml`
`theme`	`<theme>/i18n.yaml`
`target`	`<theme>/<target>/i18n.yaml`
`project`	`i18n.yaml` placed next to your `markpublish.yaml`

Green highlights mark layers that define custom values compared to the package default – the `--overridden` flag limits the output to these rows. Rows that would cause a build failure always remain visible: a filter that hides fatal errors would defeat diagnostic safety.

Language codes appear in parentheses only when they deviate from the active document language: `(*)` denotes a language-independent key, while `(en)` indicates a fallback language resolution. At the bottom of the output, full paths to all four cascade layers are displayed along with their availability status.

Pre-Build Signature Validation and Summary:
`markpublish labels` also validates the theme's Typst template directly against the parameter call markpublish will execute during rendering. If the theme fails to declare required parameters (such as `meta: (:)`), `labels` reports the signature error immediately. A summary at the bottom of the output provides a clean tally of fatal errors, warnings, and unused fields.

3.8 Global Options and Environment Variables

The following options and environment variables control markpublish globally:

Option / Variable	Description
`--version`, `-v`	Prints the installed version of markpublish and exits.
`--help`	Displays built-in CLI help and option summaries in the terminal.
`--ui-lang`	Interface language for terminal output during this invocation (`en`, `de`).
`MARKPUBLISH_UI_LANG`	Environment variable to permanently set the terminal interface language.
`MARKPUBLISH_TEMPLATES_DIR`	Environment variable pointing to a global custom templates directory.

markpublish resolves the user interface language in the following order of precedence:

1. The `--ui-lang` CLI option
2. The `MARKPUBLISH_UI_LANG` environment variable
3. The operating system's UI language (Windows display language or POSIX variables `LC_ALL`, `LANG`)
4. English (`en`) as the default fallback

(Note: The interface language controls only terminal messages and CLI help. The typesetting language, hyphenation, and date formatting of the generated PDF document are controlled completely independently by `language:` in `markpublish.yaml`.)

```
# Force terminal output to English for this command
markpublish --ui-lang en build
```

CHAPTER 4

Project Configuration with markpublish.yaml

> Structuring documents into Document, Parts, and Chapters with practical examples.

AT A GLANCE

The central control file for every markpublish project is `markpublish.yaml`. It configures the document declaratively: metadata, layout specifications, structural hierarchy, and the mapping of Markdown files to chapters and parts.

4.1 The Three Structural Tiers

markpublish organizes documents across three clearly defined levels:

```
document
└── parts (Sections)
    └── chapters (Chapters)
```

The configuration goes no deeper than this. Within a chapter, the Markdown file itself carries the structure, using `##`, `###` and so on.

4.1.1 Document (Document Level)

The top-level `document:` key describes the document as a whole:

Metadata Title, subtitle, authors, version, date, language, status, and copyright notices.

Layout Toggles Cover page display (`cover`), header activation (`header`), and footer activation (`footer`).

Global Table of Contents Settings Depth controls for the global table of contents (`document_toc`), part-level tables of contents (`part_toc`), and local chapter tables of contents (`chapter_toc`).

> **No Booleans in TOC Switches**
> The switches `document_toc`, `part_toc`, and `chapter_toc` do **not** accept Booleans (`true` or `false`). Use `"full"` (full depth), `"none"` (no TOC), or a positive integer starting at `1` for the maximum heading depth (e.g., `2`). Writing `document_toc: false` aborts validation with a clear error message.

Numbering Rules Settings for automatic numbering (`autonum_pattern`, `autonum_reset`).

4.1.1.1 Custom Metadata Fields

Under `document:`, **arbitrary custom fields** are permitted beyond the standard keys. They are forwarded to the theme, but are not printed automatically:

```yaml
document:
  title: "Project Report"
  department: "Controlling"
```

```
  client: "Acme Corporation"
  reviewed: true
```

The metadata block on the cover page only displays fields that the active theme specifically arranges – in the built-in standard theme, these are version, date, author, copyright, and status. A custom field is only displayed when a theme explicitly requests it (see chapter *Theme System and Internationalization*).

> **Why not automatically?**
> If every arbitrary field landed unprompted on the cover page, so would every typo: `departmnet: "Controlling"` would create an unwanted line on the title page. Controlling the title page layout remains the theme's responsibility.

Three essential things to keep in mind:

- **Unused fields are reported by `markpublish labels`.** Any field not referenced by a theme appears with the status *unused* and is listed in the summary at the bottom – this is exactly where mistyped keys stand out. Checking `labels` before your first build saves time.
- **Labels come from the i18n cascade.** When a theme prints the field, it needs a label string; if none is found, the cover page prints the key itself – e.g. `department` instead of `Department`. To define labels, create an `i18n.yaml` file next to your `markpublish.yaml`:

```
# i18n.yaml in the project root
en:
  department: "Department"
  client: "Client"
  reviewed: "Reviewed"
```

This file is the final stage of the cascade and overrides both application and theme defaults (see chapter *Theme System and Internationalization*).

- **Types are preserved.** A `true` is a boolean value, not raw text: the cover page typesets `Yes` or `No` (or `Ja` / `Nein`) depending on the document language. These strings are looked up as `bool_true` and `bool_false` in the i18n cascade.

4.1.2 Parts (Sections)

A document is organized into high-level sections via `parts:` (e.g., "Main Part", "Case Studies", "Appendices"):

- A document must always be structured using `parts:`; a flat `chapters:` list at the top level is rejected.
- A part groups logically related chapters together.

- By default, a part produces no separate page (`break_before: "none"`): it groups its chapters and passes down its settings, but occupies no page and does not appear in the table of contents. With `break_before: "divider"`, it receives a styled divider page; with `break_before: "page"`, it receives a heading on a fresh page.
- Settings defined at the part level (such as TOC depth or an `autonum_pattern`) are automatically inherited by all chapters contained within.
- Under `parts:`, only declared configuration keys are allowed. Unknown keys (such as `break_befor`) are rejected with fuzzy suggestions to prevent unnoticed misconfigurations.

4.1.3 Chapters (Chapters)

The `chapters:` list within a part references the actual Markdown content files:

- Each chapter points to its Markdown file (`file:`). Authors can write their Markdown files naturally starting with a `#` heading.
- **`show_title:`** (`true` or `false`, default: `true`) controls whether the file's `#` heading should be rendered on the content page. If a divider page precedes the chapter (`break_before: "divider"`), setting `show_title: false` prevents the title from repeating on the next page – the content page then starts directly with the introductory text or the first subheading.
- Optional keys **`toc_title:`** (for tables of contents and headers) and **`divider_title:`** (for divider pages) allow defining distinct alternative titles (e.g., a concise short form in the table of contents versus an extensive title on the chapter page). When omitted, both inherit the file's `#` heading.
- When generating a divider page (`break_before: "divider"`) or listing the chapter in the table of contents, at least one `#` heading or the corresponding title key (`divider_title` / `toc_title`) must exist; otherwise, the build aborts with an informative error message.
- Chapters can control via `break_before` whether a divider page is generated, a simple page break occurs, or content flows seamlessly.

4.2 Example of a Complete Configuration

The following example demonstrates a realistic, complete `markpublish.yaml`:

```yaml
# markpublish.yaml

document:
  title: "Annual Project Report"
  subtitle: "Analysis, Results, and Outlook"
  summary: "Comprehensive report covering the project phases of the past fiscal
year."
  author: "Controlling Working Group"
```

```yaml
  date: "auto"
  version: "1.2.0"
  language: "en"
  copyright: "© 2026 Acme Corporation"

  # Layout elements
  cover: true
  header: true
  footer: true

  # TOC depths
  document_toc: 2
  part_toc: 2
  chapter_toc: "none"

  # Numbering: slot 1 is the part, slot 2 the chapter, slot 3 the H2
  autonum_pattern: "_|1|.1|+"

theme: "default"

parts:
  - part: "Main Part"
    break_before: "none"
    chapters:
      - file: "chapters/01_introduction.md"
        break_before: "page"

      - file: "chapters/02_analysis.md"
        # Concise title for TOC when file H1 is long:
        toc_title: "Market Analysis"
        break_before: "page"

      - file: "chapters/03_results.md"
        break_before: "divider"

  - part: "Appendices"
    break_before: "divider"
    # On a part, slot 1 is the chapter: '_' leaves it unnumbered and the
    # H2 below start at 1 - afresh in every chapter.
    autonum_pattern: "_|1|.1|+"
    autonum_reset: true
    chapters:
      - file: "chapters/appendix_tables.md"

      - file: "chapters/appendix_glossary.md"
```

A complete reference of all available keys, data types, default values, and valid combinations for each of the three tiers can be found in **Appendix A: markpublish.yaml Schema Specification** at the end of this manual.

CHAPTER 5

Theme System and Internationalization

Theme structure, three-tier resolution hierarchy, and text cascade (i18n).

AT A GLANCE

markpublish strictly separates content from presentation: text content is written in Markdown, while the visual appearance, typography, and page layouts are controlled via Typst templates. This architecture is complemented by a flexible cascade for internationalization (i18n).

5.1 Theme Resolution Hierarchy

When markpublish searches for a theme (specified via the `--theme` CLI flag, the `theme:` configuration setting, or the `default` fallback), the resolver searches through the following locations in order (the first match wins):

1. **User Templates (`user`):**
   Templates in the current user's home directory (`~/.markpublish/templates/<theme>/`) or platform-specific AppData directory. This allows author-wide default templates across all projects.
2. **Project / Common Directory (`common`):**
   Templates in the project directory (`./templates/<theme>/`) or in a directory explicitly configured via `templates_dir:` in `markpublish.yaml`, the `--templates-dir` CLI flag, or the `MARKPUBLISH_TEMPLATES_DIR` environment variable.
3. **Package Templates (`package`):**
   The built-in default theme (`default`). It serves as a reliable fallback whenever no template is found at the user or project level.

5.2 Theme Structure

A complete markpublish theme has the following directory layout:

```
templates/
└── my-theme/
    ├── i18n.yaml            # Cross-format theme text labels
    └── pdf/
        ├── template.typ     # Typst layout template for PDF
        ├── i18n.yaml        # Format-specific theme text labels
        ├── fonts/           # Optional font files (e.g., .ttf, .otf)
        └── assets/          # Static graphics, logos, and icons
            └── icons/
```

5.2.1 The Theme Contract (`setup-document` and `meta`)

The core of `template.typ` is the Typst function `setup-document`. It receives layout switches, text labels, and all document metadata from markpublish:

```
#let setup-document(
  language: "en",
  show-cover: true,
  show-toc: true,
  toc-title: "Table of Contents",
  toc-depth: 3,
  show-header: true,
  show-footer: true,
  meta: (:),
  labels: (:),
  body,
) = {
  // ...
}
```

> **Metadata Delivery via `meta`**
> markpublish delivers all document metadata packaged as a structured Typst dictionary `meta: (:)`. Every theme must declare this parameter in `setup-document` (or accept `..rest`). If the parameter is missing, `markpublish labels` flags it as a signature mismatch error before the build begins.

Inside the Typst template, access individual metadata fields by key:

```
let title = meta.at("title").value
let subtitle = meta.at("subtitle").value
let version = meta.at("version").value
```

Custom or optional metadata fields should always be queried with a safe fallback:

```
let department = meta.at("department", default: (value: none)).value
```

If a theme queries a field without `default:` that is not defined in the configuration, compilation aborts with an informative `UndefinedMetadataError`. The error cites the exact file line in the theme and references `markpublish.yaml`.

5.2.2 What Appears on the Title Page (`cover-fields`)

The metadata grid on the cover page does not automatically display everything under `document:`; it shows precisely the fields designated by the theme – in the built-in standard theme, these are version, date, author, copyright, and status. Title, subtitle, and summary are typeset as distinct header blocks above.

Which fields appear on the cover page is determined inside the theme by the `cover-fields` function:

```
#let cover-fields(meta) = (
  meta.at("version", default: none),
  meta.at("date", default: none),
  meta.at("author", default: none),
  meta.at("copyright", default: none),
  meta.at("status", default: none),
)
```

To display `department:` or `client:` on the cover page, export the theme (see below) and add a line to `cover-fields`. The accompanying label is looked up in the project's `i18n.yaml`.

5.2.3 Typed Metadata Values

Metadata preserves its native YAML data type on its way to Typst: * Integers, floats, and strings remain numbers and strings. * `reviewed: true` arrives as a native Typst boolean. The standard theme formats this automatically using the label keys `bool_true` ("Yes") and `bool_false` ("No") from the i18n cascade. * Lists are formatted cleanly with comma separation.

Which fields a theme actually consumes can be inspected with `markpublish labels`: any field ignored by the theme is flagged as *unused*.

5.3 Creating and Customizing Themes

The fastest and safest way to develop your own corporate design is to export the built-in standard theme:

```
markpublish export-template default ./templates --target pdf
```

This command copies the complete default theme into the local directory `templates/default/`. You can then edit the files directly, customize fonts, adjust color palettes, or embed your company logo. markpublish automatically uses your customized project template on subsequent `build` commands.

The `template.typ` layout file is written in the Typst typesetting language and comprehensively commented in the default theme. For advanced modifications beyond colors and fonts, the official language documentation is available at https://typst.app/docs.

5.4 Internationalization and the Text Cascade (i18n)

Professional documents require various static text strings that do not originate from chapter Markdown files but are generated by the template – such as “Table of Contents”, “Chapter”, “Page X of Y”, or labels on the title page.

markpublish manages these strings through a cascading system of `i18n.yaml` files.

5.4.1 The Four-Tier Label Cascade

When resolving a text label (e.g., `toc_title`), markpublish traverses the following layers in order – later entries override earlier ones:

1. **Package Base (`markpublish/i18n.yaml`):** Comprehensive default strings for all supported languages.
2. **Theme Level (`<theme>/i18n.yaml`):** Themes can supply their own terminology or design phrasing.
3. **Format Level (`<theme>/pdf/i18n.yaml`):** Format-specific text customizations for the theme.
4. **Project Level (`i18n.yaml` next to `markpublish.yaml`):** Document-specific texts with highest priority.

The project level is where custom metadata fields are labeled: if you specify `department: "R&D"` under `document:`, write `department: "Department"` into the project’s `i18n.yaml`. Without this entry, the cover page falls back to printing the raw key name. You can also override any theme text for an individual project without modifying the theme itself.

5.4.2 Structure of i18n.yaml

An `i18n.yaml` file contains translations grouped by language code:

```yaml
en:
  toc_title: "Table of Contents"
  chapter: "Chapter"
  part: "Part"
  page: "Page"
  page_of: "of"
  version: "Version"
  author: "Author"

de:
  toc_title: "Inhaltsverzeichnis"
  chapter: "Kapitel"
  part: "Abschnitt"
  page: "Seite"
  page_of: "von"
```

```
  version: "Version"
  author: "Autor"
```

5.4.3 Language Detection and Validation

The document language is set in `markpublish.yaml` via the `language:` key (e.g., `language: "en"`). If omitted, markpublish automatically detects your operating system language.

You can run `markpublish labels` at any time to inspect all active text labels for your project and verify which file provided each string.

CHAPTER 6

Advanced Markdown Features

> Markup and typographical presentation of callouts, definition lists, task lists, and mathematical formulas.

AT A GLANCE

markpublish supports the full range of modern CommonMark and GitHub Flavored Markdown standards. Fundamental elements such as headings, paragraphs, text formatting (*italic*, **bold**), tables, bullet lists, code blocks, and hyperlinks work exactly as expected.

This chapter focuses exclusively on features and typographic enhancements beyond the standard supported by markpublish. Each section demonstrates first the Markdown syntax and immediately below the rendered result as typeset in this manual.

6.1 Callout Boxes (Admonitions)

To emphasize important passages, markpublish supports GitHub-compatible callout blocks. They are written as blockquotes whose first line specifies the box type in square brackets:

```
> [!NOTE]
> General information with helpful background details.

> [!TIP]
> A practical tip to streamline your workflow.

> [!IMPORTANT]
> Important information essential for understanding.

> [!WARNING]
> Warning against potential misconfigurations or unexpected behavior.

> [!CAUTION]
> Critical safety note to prevent potential data loss.
```

In the PDF, these are rendered as styled boxes with colored borders and icons:

> **Note**
> General information with helpful background details.

> **Tip**
> A practical tip to streamline your workflow.

> **Important**
> Important information essential for understanding.

> **Warning**
> Warning against potential misconfigurations or unexpected behavior.

> **Caution**
> Critical safety note to prevent potential data loss.

If a custom title is desired, it can be added directly after the type on the first line:

```
> [!NOTE] Custom Title
> Dedicated content with a customized header.
```

> **Custom Title**
> Dedicated content with a customized header.

6.1.1 Automatic Localization via i18n Labels

Without a custom title, the header of the callout box (“Note”, “Tip”, “Important”, “Warning”, “Caution”) is automatically populated via the `i18n.yaml` file for the active document language:

Box Type	i18n Key Used	English Label
`[!NOTE]`	`alert_note`	Note
`[!TIP]`	`alert_tip`	Tip
`[!IMPORTANT]`	`alert_important`	Important
`[!WARNING]`	`alert_warning`	Warning
`[!CAUTION]`	`alert_caution`	Caution

6.1.2 Alternative Syntax with `!!!`

Alongside the GitHub notation, markpublish also supports the admonition syntax popularized by MkDocs. The type follows three exclamation marks, an optional title is written in quotes, and content is indented by four spaces:

```
!!! warning "Custom Title"
    Indented content can span multiple paragraphs.
```

> ⚠ **Custom Title**
> Indented content can span multiple paragraphs.

Both notations generate identical styled boxes. One minor difference remains: without a custom title, `!!!` uses the extension's English default ("Note", "Tip", "Warning") rather than looking up the label from the i18n cascade. To use the localized title header, either write `> [!NOTE]` or supply an explicitly empty title string:

```
!!! note ""
    The title bar is then retrieved from the i18n cascade.
```

> ⓘ **Note**
> The title bar is then retrieved from the i18n cascade.

An unrecognized box type (such as `!!! info`) is rendered as a Note box, retaining its name as the title.

6.2 Definition Lists

For glossaries, parameter listings, or terminology overviews, definition lists provide clean typographical structure. The term stands on its own line, followed by the explanation indented with a colon:

```
Markdown
: Simple, readable markup language for structured documents in plain text.

markpublish.yaml
: Central project configuration file controlling document hierarchy, metadata,
and numbering.

Theme
: Design template composed of Typst layouts and text labels that define visual
styling.

Typst
: Modern, high-performance typesetting engine used by markpublish to generate
print-ready PDFs.
```

This renders as a distinct block featuring highlighted terms with indented definitions:

Markdown Simple, readable markup language for structured documents in plain text.

markpublish.yaml Central project configuration file controlling document hierarchy, metadata, and numbering.

Theme Design template composed of Typst layouts and text labels that define visual styling.

Typst Modern, high-performance typesetting engine used by markpublish to generate print-ready PDFs.

6.3 Task Lists

For checklists, actionable steps, or release milestones, task lists are supported. They are written as bullet items starting with brackets:

```
- [x] Install Python 3.10 or newer
- [x] Install markpublish
- [ ] Publish your first document
```

markpublish typesets task lists cleanly without bullet points, aligning multi-line descriptions directly beneath the first line:

- [x] Install Python 3.10 or newer
- [x] Install markpublish
- [ ] Publish your first document

6.4 Footnotes

markpublish supports classic Markdown footnote syntax. This allows referencing sources or providing supplementary commentary without disrupting the flow of the main text.

Footnotes use a two-part notation: insert a numeric reference like `[^1]` or an identifier like `[^note]` within the text. The corresponding definition can be placed anywhere in the Markdown document – typically at the end of the chapter:

```
This sentence contains a numbered footnote[^1] and a reference with a text
key[^note].

[^1]: This is the body of the first footnote.
[^note]: Text keys are automatically converted into sequential numbers during
typesetting.
```

In the rendered document, this produces the following references; their text appears at the bottom of the page, with bi-directional clickable links between the reference marker and the footnote text:

This sentence contains a numbered footnote[1] and a reference with a text key[2].

6.5 Mathematical Formulas

markpublish allows inserting mathematical formulas directly inline or as display blocks. Formulas are rendered with typographic precision by Typst.

Common mathematical expressions such as fractions (`\frac{a}{b}`), square roots (`\sqrt{x}`), summations (`\sum`), integrals (`\int`), Greek characters (`\alpha`, `\pi`, `\sigma`), and native Typst math expressions are supported. Inline formulas are enclosed by single dollar signs, display blocks by double dollar signs:

```
The famous mass-energy equivalence is $E = m c^2$.

Circular area calculation and quadratic formula:

$$ A = \pi \cdot r^2 \quad \text{and} \quad x = \frac{-b \pm \sqrt{b^2 - 4 a
c}}{2 a} $$
```

In the PDF, this is typeset as:

The famous mass-energy equivalence is $E = mc^2$.

Circular area calculation and quadratic formula:

$$A = \pi \cdot r^2 \text{ and } x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Typographical note on variables:
In Typst Math, multi-letter words denote function names or identifiers (such as `sin`, `cos`, `sqrt`). Multiplication of individual variables is therefore written with space separation (e.g., `m c^2` or `4 a c`).

6.6 Typographical Text Formats and Punctuation

Beyond standard Markdown, markpublish supports fine typographical formatting and authentic punctuation glyphs:

[1] This is the body of the first footnote.
[2] Text keys are automatically converted into sequential numbers during typesetting.

Formatting	Input	Output	Typical Use Case
Superscript	`10^3^` or `m^2^`	10^3 or m^2	Powers, square/cubic meters
Subscript	`H~2~O` or `CO~2~`	H_2O or CO_2	Chemical formulas, indices
Strikethrough	`~~outdated~~`	~~outdated~~	Corrections, deprecated items
En-Dash	`10--20 hours`	10–20 hours	Date, time, and page ranges
Em-Dash	`Thought --- or not`	Thought — or not	Parenthetical thoughts, narrative pauses
Ellipsis	`Loading...`	Loading…	Omissions, trailing off

Prose vs. Mathematical Typesetting

The syntax `^superscript^` and `~subscript~` is designed for standard units (m^2) and chemical formulas (H_2O) within text. For mathematical formulas and equation systems, use the dedicated math environment with `$...$` or `$$...$$` (see section *Mathematical Formulas*).

6.7 Automatic Symbols and Arrows

Frequent symbols and operators typed in plain text are automatically converted into proper typographical glyphs:

Symbol	Input	Output	Typical Use Case
Arrows	`-->`, `<--`, `<-->`	→, ←, ↔	Flow steps, back-references, equivalence
Fractions	`1/2`, `1/4`, `3/4`	½, ¼, ¾	Measurements, fractional values
Arithmetic Glyphs	`+/-`, `=/=`	±, ≠	Tolerances (±), inequality (≠)
Copyright & Trademarks	`(c)`, `(tm)`, `(r)`	©, ™, ®	Copyright, Trademark, Registered

6.8 Automatic Linking and Cross-References

Web URLs and email addresses are automatically recognized and turned into clickable links. In addition, internal cross-references allow linking directly to any heading:

Feature	Input	Output	Description
Web Address (Magic Link)	`https://example.com`	https://example.com	URLs are linked without explicit Markdown brackets
Email Address (Magic Link)	`contact@example.com`	contact@example.com	Email addresses become clickable `mailto:` links
Document Cross-Reference	`[Chapter top](#advanced-markdown-features)`	Chapter top	Clickable internal link pointing to a heading anchor

6.9 Escape-Free Special Characters

Many documentation systems require cumbersome escaping for common characters. markpublish processes standard symbols in plain text safely without collision:

Category	Example	Behavior in Document
Currency Amounts	`$5.00` or `10.50 $`	Not misparsed as mathematical formulas.
Programming Languages	`C#`	Does not trigger internal Typst commands.
Usernames & Mentions	`@author` or `user@domain.com`	Printed safely as text or autolinked.
Technical Identifiers	`<target_directory>`	Angle brackets remain intact as literal text.

SECTION

Appendices

Complete specification tables and troubleshooting guidance.

CONTENTS OF THIS SECTION

Appendix A: markpublish.yaml Schema Specification

This appendix provides the complete specification of all configuration options supported in `markpublish.yaml`.

A.1 Document Level (document)

The `document:` section defines global metadata, layout switches, table of contents depths, and numbering styles for the entire document.

Key	Type	Default	Description
`title`	String	*(Required)*	Document title (appears on cover page, header, and metadata).
`subtitle`	String	`None`	Document subtitle.
`summary`	String	`None`	Executive summary or abstract (printed on cover page).
`author`	String	`None`	Author or organization name.
`status`	String	`None`	Document lifecycle status (e.g., “Draft”, “Approved”, “Confidential”).
`copyright`	String	`None`	Copyright notice (e.g., “© 2026 Frank Winter”).
`date`	String	`"auto"`	Document date. Set to `"auto"` or `"today"` to use current date.
`version`	String	`None`	Version identifier (e.g., `"2.0.0"`). Only printed when explicitly set.
`language`	String	*(System language)*	ISO language code (e.g., `"en"` or `"de"`). Governs hyphenation and UI strings.
`cover`	Boolean	`false`	Controls whether a styled cover page is generated.
`header`	Boolean	`true`	Enables or disables running headers throughout the document.

Key	Type	Default	Description
`footer`	Boolean	`true`	Enables or disables running footers (including page numbering).
`document_toc`	Scope	`"full"`	Depth for the main table of contents (`"none"`, `"full"`, or integer ≥ 1).
`part_toc`	Scope	`"full"`	Default depth for local tables of contents on part divider pages.
`chapter_toc`	Scope	`"full"`	Default depth for local tables of contents on chapter divider pages.
`autonum_pattern`	String	*see below*	How numbers are built; slot 1 is the part, slot 2 the chapter. `none` switches numbering off.
`autonum_reset`	Boolean	`false`	Makes the levels below start again at one when this block is entered.
`part_label`	String	from i18n	Word naming a part (e.g. `"Part"`). Notated empty (`""`), it is dropped.
`chapter_label`	String	from i18n	Word naming a chapter (e.g. `"Chapter"`). Notated empty (`""`), it is dropped.
`pagenum_reset`	Boolean	`false`	Restarts page numbering at page 1 for each part or chapter.

> **Custom Metadata Fields under `document:`**
> Beyond standard fields, arbitrary custom metadata fields can be added under `document:` (e.g., `department: "R&D"`, `reviewed: true`). They are delivered typed into the theme (`meta: (:)`).
>
> Static text translations (`document.i18n` or `document.labels`) are not permitted here and cause validation to fail – localization belongs exclusively in `i18n.yaml` files within the cascade.

A.2 Global Project Settings

Directly at the top level of `markpublish.yaml` (alongside `document:` and `parts:`), the following settings can be configured:

Key	Type	Default	Description
`theme`	String	`"default"`	Name of the active theme.
`templates_dir`	String	`None`	Optional custom directory path containing theme templates.

> **Strict Top-Level Key Validation**
> At the top level of `markpublish.yaml`, only declared configuration keys (`document`, `theme`, `templates_dir`, `parts`) are permitted. Unknown keys (such as typos like `theam:`) are strictly rejected with close-match suggestions.

A.2.1 Numbering Patterns

`autonum_pattern` describes how numbers are built, as a chain of slots separated by `|`. Slot 1 belongs to the first level *below* the place the pattern stands: under `document:` that is the part, on a part the chapter, on a chapter the H2. A pattern therefore never describes the level it is written at.

Symbol	Result
`1`	1, 2, 3 …
`01`, `001`	01, 02 … or 001, 002 … (fixed width)
`a` / `A`	a, b, c … / A, B, C …
`i` / `I`	i, ii, iii … / I, II, III …
`_`	level stays unnumbered
`+`	repeats the slot before it for every deeper level

Everything else in a slot is a literal and needs no quoting; only a reserved character meant literally goes into single quotes (`'Article '1`). A malformed pattern aborts the build.

```
"_|1|.1|+"        part unnumbered, chapter 1, section 1.1   (default)
"I|1|.1|+"        part I, II … chapters count straight through
"'Appendix 'A|.1" Appendix A, Appendix A.1
```

A notated `label` puts the word in front of the number of the heading and its table-of-contents entry: `A` becomes `Appendix A:`. The subheadings are untouched and keep counting `A.1`, `A.2` — which is why the word belongs in this key and not in the pattern. The separator comes from the i18n cascade (`label_separator`).

The part number does not enter the chapter numbers: it appears on the divider page and in the table of contents, not in front of every chapter. A part with `document_toc: "none"` gets no number and consumes none.

A.3 Section Level (parts)

The `parts:` list subdivides the document into high-level sections. A part groups chapters together and passes down its settings.

Key	Type	Default	Description
`part`	String	*(Required)*	Section name (e.g., "Main Part", "Appendices").
`toc_title`	String	`None`	Optional title override for TOC and headers (default: `part`).
`divider_title`	String	`None`	Optional title override on the section divider page (default: `part`).
`subtitle`	String	`None`	Section subtitle (appears on divider page).
`summary`	String	`None`	Section summary (appears on divider page).
`break_before`	String	`"none"`	Break behavior: `"divider"` (divider page), `"page"` (heading on new page), or `"none"` (structural grouping only, no dedicated page).
`document_toc`	Scope	`None`	Overrides this section's contribution to the main TOC.
`part_toc`	Scope	`None`	Controls the local table of contents on the section divider page.
`chapter_toc`	Scope	`None`	Propagates chapter TOC settings down to all chapters in this part.
`autonum_pattern`	String	`None`	Numbers for this part; slot 1 is the chapter here.
`autonum_reset`	Boolean	`None`	Makes the chapters of this part start again at one.
`label`	String	from i18n	Word naming this part. Notated empty (`""`), it is dropped.
`chapter_label`	String	inherited	Word for every chapter of this part (e.g. `"Appendix"`). Notated empty (`""`), it is dropped.
`pagenum_reset`	Boolean	`None`	Resets page numbering to 1 at the start of this section.

Key	Type	Default	Description
`chapters`	List	*(Required)*	List of content chapters in this part (at least 1 entry).

A.4 Chapter Level (chapters)

The `chapters:` list specifies the content files. Chapters always sit directly under an entry of `parts:`.

The chapter heading on the content page is determined by default by the file's leading `#` heading (suppressible via `show_title: false`). For tables of contents and divider pages, two explicit title overrides are available:

Key	Type	Default	Description
`file`	String	`None`	Relative path to the Markdown file (e.g., `"chapters/01_intro.md"`).
`show_title`	Boolean	`true`	Controls whether the file's `#` heading is printed on the content page.
`toc_title`	String	`None`	Title for tables of contents and headers. Default: file H1.
`divider_title`	String	`None`	Title on chapter divider pages (`break_before: "divider"`). Default: file H1.
`subtitle`	String	`None`	Chapter subtitle (appears on divider pages).
`summary`	String	`None`	Short summary (used on divider pages).
`break_before`	String	`"page"`	Page break behavior: `"page"` (new page), `"divider"` (divider page), or `"none"`.
`document_toc`	Scope	`None`	Contribution of this chapter to the main TOC (`"none"`, `"full"`, integer).
`chapter_toc`	Scope	`None`	Local table of contents on this chapter's divider page.
`autonum_pattern`	String	`None`	Numbers inside this chapter; slot 1 is the H2 here. The chapter's own number comes from its part.
`autonum_reset`	Boolean	`None`	Makes the headings of this chapter start again at one.

Key	Type	Default	Description
`label`	String	inherited	Word naming this chapter (e.g. `"Excursus"`). Notated empty (`""`), it is dropped.
`pagenum_reset`	Boolean	`None`	Resets page numbering to 1 at the start of this chapter.

> **Important**
> Under `parts:` and `chapters:`, **only** the keys listed above are allowed. Any unrecognized key halts compilation and provides fuzzy spelling suggestions – catching mistakes like `break_befor` before chapters are rendered with incorrect breaks. Custom metadata fields belong exclusively under `document:`.

A.5 Table of Contents Scope Controls

The table of contents settings `document_toc`, `part_toc`, and `chapter_toc` accept the following values:

Value	Description
`"none"`	Disables the table of contents completely or excludes the item.
`"full"`	Includes all heading levels without depth limitation.
Positive Integer (e.g., `1`, `2`, `3`)	Limits TOC depth strictly to the specified number of heading levels.

> **No Booleans Allowed**
> Boolean values (`true` or `false`) are invalid for TOC scope switches and trigger a `ConfigurationError`. To disable a table of contents, always specify `"none"`.

Appendix B: Troubleshooting and FAQ

This appendix offers actionable guidance for typical issues during the publishing process along with answers to frequently asked questions.

B.1 Diagnosing Compilation Errors

If a compilation run aborts due to an error from the Typst engine, markpublish automatically saves the fully generated Typst source code in your working directory:

```
.markpublish/last_failed_build.typ
```

B.1.1 Troubleshooting Procedure

1. Open `.markpublish/last_failed_build.typ` in any text editor.
2. Search for the keyword, variable name, or text snippet cited in the error message.
3. The surrounding context directly reveals which Markdown chapter or formatting block triggered the issue.

B.2 Common Causes and Solutions

B.2.1 Images or Graphics Not Found

Problem: Typst aborts with a message such as `image not found`.

Solution:

- Paths to images in Markdown (e.g., `![Diagram](images/architecture.png)`) are always resolved **relative to the respective Markdown file**.
- If `01_chapter.md` is located in `chapters/`, the `images/` folder must either reside in `chapters/images/` or be referenced as `../images/architecture.png`.
- markpublish automatically copies local image files to the internal build directory during compilation.

B.2.2 Invalid Indentation in Configuration (YAML)

Problem: markpublish reports `Configuration error: mapping values are not allowed here` or `YAML parsing error` on startup.

Solution:

- YAML is strict regarding indentation. Always use **two spaces** for indentation and never tabs.
- Ensure list markers (`-`) and nested keys are aligned consistently.

B.2.3 Customizing Static Text Labels (i18n)

Problem: A generated string (such as "Table of Contents" or "Chapter") should be reworded or is missing in a new language.

Solution:

- Run `markpublish labels` to display all resolved text variables and their cascade layer.
- Add or override the desired keys either in a project-local `i18n.yaml` (next to `markpublish.yaml`) or in your theme's `i18n.yaml`.

B.2.4 Fonts and Font Families

Problem: Font-related error messages or unexpected font appearance.

Solution:

- The standard theme uses the bundled **Open Sans** font family; manual font installation on your operating system is not required.
- For custom themes, you can place font files (`.ttf`, `.otf`) directly into your theme's `pdf/fonts/` folder. markpublish automatically includes theme fonts in the search path.

B.2.5 Missing Metadata Field (UndefinedMetadataError)

Problem: The build halts with the message `Metadata '...' is queried by theme but not defined in document`.

Solution:

- The theme queries a field via `meta.at("key")` that is not defined under `document:` in `markpublish.yaml` and has no fallback in the theme.
- Add the missing field under `document:` in your `markpublish.yaml` (custom fields are permitted without restriction), or define a fallback value in the theme: `meta.at("key", default: (value: none)).value`.

B.2.6 Missing Text Label (UndefinedLabelError)

Problem: The build halts with the message `Label '...' is referenced in template but not defined in any i18n layer`.

Solution:

- The template accesses a label via `labels.at("...")` that is not defined in any layer of the cascade for the active document language.
- Add the key under the appropriate language code (e.g., `en:`) in an `i18n.yaml` file in your project directory or theme.

B.2.7 Theme Signature Mismatch

Problem: `markpublish labels` or the build reports that the theme and function signature do not match.

Solution:

- The Typst function `setup-document` in `template.typ` does not declare all required parameters.
- markpublish delivers all metadata packaged in the dictionary `meta: (:)`. Ensure your theme declares this parameter: `#let setup-document(..., meta: (:), labels: (:), body) = { ... }` (or accepts `..rest`).

B.2.8 Rejection of Unknown Keys (unknown_key)

Problem: markpublish aborts with a message such as `chapters does not recognize key: 'break_befor' -- did you mean 'break_before'?`.

Solution:

- At the root level (`markpublish.yaml`) as well as section (`parts:`) and chapter (`chapters:`) levels, markpublish strictly rejects unknown keys to catch typos early. Check the reported key; custom metadata fields are exclusively permitted under `document:`.

B.2.9 Target Directory on init is Not Empty

Problem: `markpublish init` aborts with `Destination directory ... already exists and is not empty`.

Solution:

- `init` never overwrites existing files. Choose a new directory name or empty the directory before initialization.

B.3 Frequently Asked Questions (FAQ)

B.3.1 What is the difference between interface language and document language?

markpublish strictly separates the CLI language from the document typesetting language:

- **Interface language (`--ui-lang`, `MARKPUBLISH_UI_LANG`):** Controls the language of terminal error messages, help texts, and diagnostic tables (`en` or `de`). Without this flag, your system language applies.
- **Document language (`language:` in `markpublish.yaml`):** Governs typography, hyphenation, date formatting, and fixed text labels (such as "Table of Contents" or "Chapter") in the output PDF. A terminal in one language can effortlessly compile documents in another.

B.3.2 How do I prevent a divider page before a chapter?

By default, a chapter starts on a new page with `break_before: "page"`. To attach a chapter seamlessly without a page break, set `break_before: "none"` in the chapter definition.

B.3.3 Why does my section not appear anywhere in the document?

Sections (`parts:`) default to `break_before: "none"`: they group chapters, occupy no page, and do not appear in the table of contents. To make a section visible, specify:

```yaml
parts:
  - part: "Appendices"
    break_before: "divider"   # styled divider page
```

With `break_before: "page"`, the section receives a heading on a fresh page instead of a divider page.

B.3.4 How can I restart page numbering for each chapter?

Set `pagenum_reset: true` at the document or chapter level in `markpublish.yaml`. markpublish automatically resets the page number to 1 at chapter start and computes total page counts consistently.

B.3.5 Are hyperlinks exported as clickable links in the PDF?

Yes. Both internal references (from the table of contents or footnotes) and external web links (written in Markdown as `[Link text](URL)`) are generated as native, clickable PDF hyperlinks.