markpublish Kurzreferenz

Konfiguration, Daten im Text und Templates

> *Nachschlagewerk zu allen Schlüsseln der markpublish.yaml, zu Platzhaltern und Anweisungen im Text und zu den Bausteinen eines Themes.*

Version 2.4.1
Datum 13.10.2026

Autor Frank Winter

Copyright © 2026 Frank Winter

markpublish.yaml

Oberste Ebene: `document`, `theme`, `parts`, optional `templates_dir`. Alle Pfade gelten relativ zum Verzeichnis dieser Datei. Bauen mit `markpublish build [CONFIG_FILE] [OPTIONEN]`.

Aufbau

Der Aufbau hat genau zwei Gliederungsstufen. Tiefer strukturiert die Markdown-Datei selbst.

```
markpublish.yaml
├── document:            Metadaten und globale Schalter
├── theme:               Name des Themes
└── parts:               Liste der Abschnitte
    ├── part:            Name des Abschnitts
    └── chapters:        Liste der Kapitel
        └── file:        eine Markdown-Datei
            ├── # H1  Kapitelüberschrift
            ├── ## H2   Gliederung
            └── ### H3  innerhalb der Datei
```

document

Metadaten und globale Layout-Schalter. Pflichtangabe ist allein `title`.

Schlüssel	Standard	Bedeutung
`title`	*Pflichtangabe*	Titel des Dokuments (Deckblatt, Kopfzeile, Dateiname)
`subtitle`	–	Untertitel auf dem Deckblatt
`summary`	–	Zusammenfassung / Abstract auf dem Deckblatt
`author`	–	Name des Autors
`date`	`auto`	Datum (`auto` oder `today` für das Baudatum)
`version`	–	Versionsnummer auf Deckblatt und in der Fußzeile
`status`	–	Dokumentstatus (z. B. `Entwurf`, `Freigegeben`)
`copyright`	–	Copyright-Hinweis auf dem Deckblatt

© 2026 Frank Winter

Schlüssel	Standard	Bedeutung
`language`	Systemsprache	ISO-Sprachcode für statische Beschriftungen (`de`, `en`)
`custom`	–	Eigene Werte für Platzhalter im Text (`{{custom.a.b}}`), auch verschachtelt
`cover`	`false`	Deckblatt erzeugen
`header` / `footer`	`true`	Kopf- und Fußzeilen aktivieren
`document_toc`	`full`	Haupt-Inhaltsverzeichnis: `none`, `full` oder Tiefe als Zahl
`part_toc`	`full`	Vorgabe für Part-Trennseiten: `none`, `full` oder Tiefe
`chapter_toc`	`full`	Vorgabe für Kapitel-Trennseiten: `none`, `full` oder Tiefe
`autonum_pattern`	*s. Nummerierung*	Aufbau der Nummern; Slot 1 ist hier der Part
`autonum_reset`	`false`	Ebenen darunter beim Betreten wieder bei 1 beginnen
`part_label`	aus i18n	Wort vor der Part-Nummer (`Teil`); `""` lässt es weg
`chapter_label`	aus i18n	Wort vor der Kapitelnummer (`Kapitel`); `""` lässt es weg
`pagenum_reset`	`false`	Seitennummerierung je Part oder Kapitel neu beginnen

Besteht das Dokument aus **genau einem Kapitel**, lässt das Haupt-Inhaltsverzeichnis dessen eigene Zeile weg: sie wiederholte nur den Titel des Deckblatts, und ein Eintrag ohne Geschwister unterscheidet nichts. Die Unterüberschriften des Kapitels bleiben aufgeführt, seine Nummer steht weiterhin über dem Text bzw. auf der Trennseite. `document_toc` zählt dabei weiter Gliederungsebenen und nicht sichtbare Zeilen: für zwei sichtbare Ebenen (H2 und H3) notieren Sie `document_toc: 3`.

Eigene Zusatzfelder unter `document:` (`abteilung: "F&E"`) erreichen das Theme über das Wörterbuch `meta` (`meta.at("abteilung").value`). Statische Beschriftungen gehören in die `i18n.yaml`, nicht hierher.

Werte unter `custom` erreichen das Theme nicht; im Markdown setzt sie `{{custom.a.b}}` ein (siehe „Daten im Text“).

`schlüssel: !file "pfad-oder-url"` lädt einen Wert aus einer anderen Datei, überall in der Konfiguration. YAML-Dateien liefern einen verschachtelten Wert, jede andere Datei ihren Text. Relative Pfade gelten ab der verweisenden Datei.

© 2026 Frank Winter

parts

Ein Part klammert Kapitel und gibt Einstellungen an sie weiter. Ohne `break_before` zeigt er sich selbst nicht.

Schlüssel	Standard	Bedeutung
`part`	*Pflichtangabe*	Name des Abschnitts
`chapters`	*Pflichtangabe*	Liste der Kapitel dieses Abschnitts
`toc_title`	`part`	Titel im Haupt-Inhaltsverzeichnis
`divider_title`	`part`	Titel auf der Trennseite
`subtitle` / `summary`	–	Untertitel und Kurzbeschreibung auf der Trennseite
`break_before`	`none`	`divider` (Trennseite), `page` oder `none`
`document_toc`	geerbt	Beitrag zum Haupt-Inhaltsverzeichnis; `none` versteckt den Part
`part_toc`	geerbt	Lokales Verzeichnis auf der eigenen Trennseite
`chapter_toc`	geerbt	Vorgabe für die Trennseiten seiner Kapitel
`autonum_pattern`	geerbt	Nummern seiner Kapitel; Slot 1 ist hier das Kapitel
`autonum_reset`	geerbt	Kapitel dieses Parts wieder bei 1 beginnen
`label`	aus i18n	Wort vor der eigenen Nummer
`chapter_label`	geerbt	Wort vor der Nummer jedes seiner Kapitel (`Anhang`)
`pagenum_reset`	geerbt	Seitennummerierung am Part-Anfang zurücksetzen

chapters

Ein Kapitel ist eine Markdown-Datei. Sein Name kommt aus deren `#`-Überschrift.

Schlüssel	Standard	Bedeutung
`file`	*Pflichtangabe*	Pfad zur Markdown-Datei
`list_of`	–	Statt `file`: Verzeichnis der `figures` oder `tables` an dieser Stelle
`show_title`	`true`	Datei-H1 auf der Inhaltsseite anzeigen

© 2026 Frank Winter

Schlüssel	Standard	Bedeutung
`toc_title`	Datei-H1	Titel im Inhaltsverzeichnis und in der Kopfzeile
`divider_title`	Datei-H1	Titel auf der Trennseite
`subtitle` / `summary`	–	Untertitel und Kurzbeschreibung auf der Trennseite
`break_before`	`page`	`divider` (Trennseite), `page` oder `none`
`document_toc`	geerbt	Beitrag zum Haupt-Inhaltsverzeichnis (`none`, `full`, Tiefe)
`chapter_toc`	geerbt	Lokales Verzeichnis auf der eigenen Trennseite
`autonum_pattern`	geerbt	Nummern *innerhalb* des Kapitels; Slot 1 ist hier die H2
`autonum_reset`	geerbt	Überschriften dieses Kapitels wieder bei 1 beginnen
`label`	geerbt	Wort vor der eigenen Nummer (`Anhang A: …`)
`pagenum_reset`	geerbt	Seitennummerierung am Kapitelanfang zurücksetzen

Vererbung

Was mit *geerbt* markiert ist, fließt von außen nach innen; die innerste Angabe gewinnt.

```
document ──────────────► parts[] ──────────────► chapters[]
  autonum_pattern          autonum_pattern        autonum_pattern
  autonum_reset            autonum_reset          autonum_reset
  document_toc             document_toc           document_toc
  part_toc                 part_toc                    –
  chapter_toc              chapter_toc            chapter_toc
  pagenum_reset            pagenum_reset          pagenum_reset
  part_label               label                       –
  chapter_label            chapter_label          label
```

Nummerierung

Ein Pattern beschreibt die Ebenen *unterhalb* der Stelle, an der es steht — Slot 1 verschiebt sich also mit dem Fundort.

© 2026 Frank Winter

Ebene	unter document	an einem Part	an einem Kapitel
Part	Slot 1	-	-
Kapitel (H1)	Slot 2	Slot 1	-
H2	Slot 3	Slot 2	Slot 1
H3	Slot 4	Slot 3	Slot 2
⋮	⋮	⋮	⋮
H6	Slot 7	Slot 6	Slot 5

Symbole: `1` → 1, 2, 3 · `01` → 01, 02 · `a` / `A` → a, b / A, B · `i` / `I` → i, ii / I, II · `_` Ebene ohne Nummer · `+` wiederholt den Slot davor für alle tieferen Ebenen. Slots trennt `|`; alles andere im Slot ist Literal, Reserviertes in Hochkommas (`'Artikel '1`).

```
  an document   "_|1|.1|+"              Part ohne Nummer · Kapitel 1 · 1.1
  an document   "_|_|1|.1|+"            Ein-Kapitel-Dokument: H2 wird 1., 2.,
3.
  an document   "I|1|.1|+"              Teil I · Kapitel 1, 2, 3 durchlaufend
  am Part       "A|.1|+"                Kapitel A · A.1 · A.1.1
  am Part       "'Artikel '1|' ('1')'"  Artikel 1 · Artikel 1 (2)
```

Ohne Angabe gilt `"_|1|.1|+"`: Parts ohne Nummer, Kapitel 1, 1.1, 1.1.1.

Die Part-Nummer geht **nicht** in die Kapitelnummern ein; ein Part mit `document_toc: "none"` bekommt keine und verbraucht keine. Ein notiertes `label` tritt vor die Nummer der Überschrift (`Anhang A: Schema`), ohne die Unterüberschriften zu erreichen (`A.1`).

© 2026 Frank Winter

Daten im Text

Werte aus `document:` und `custom:` lassen sich im Markdown einsetzen, wiederholen, nachschlagen und an Bedingungen knüpfen. Ein Wert ohne Eintrag bricht den Bau mit Datei und Zeile ab.

Eingabe	Wirkung
`{{author}}`, `{{custom.a.b}}`	Wert einsetzen; Mail- und Webadressen werden Links
`{% for x in custom.gruppe %}` … `{% endfor %}`	Zeilen je Eintrag wiederholen, in YAML-Reihenfolge
`{% for (x, key) in custom.gruppe %}`	dazu den Namen des Eintrags als `{{key}}`
`{% set kurz = custom.a.b %}`	Kurzname für einen Pfad, bis Datei- bzw. Blockende
`custom.systeme[d.system]`	Nachschlagen: Wert von `d.system` als Schlüssel einsetzen
`{% if pfad %}` … `{% else %}` … `{% endif %}`	Zeilen nur bei `true`; `else` optional
`{% if pfad == "wert" %}`	Vergleich mit einem Text
`\{{author}}`, `\{% … %}`	wörtlich zeigen; in Code ohnehin

Anweisungen stehen allein in ihrer Zeile (in Zitat oder Hinweisbox mit `>` davor) und verschwinden beim Bau, eine Tabelle bleibt zusammenhängend. Nachschlagen geht nur im Markdown, nicht in den Angaben unter `document:` selbst.

```
| Datenfluss | System | Betrieb |
| :--- | :--- | :--- |
{% for d in custom.datenfluesse %}
{% set sys = custom.systeme[d.system] %}
{% if sys.extern %}
| {{d.bezeichnung}} | {{sys.name}} | Dienstleister |
{% else %}
| {{d.bezeichnung}} | {{sys.name}} | intern |
{% endif %}
{% endfor %}
```

© 2026 Frank Winter

Templates und Themes

Alles Sichtbare steckt in einem Theme: Typografie, Seitenlayout und statische Texte. Mitgeliefert wird das Standard-Theme `default` für PDF.

Wo ein Theme gesucht wird

Der erste Treffer gewinnt — ein Theme kommt immer aus genau einer Quelle, nie gemischt. Welches Theme verwendet wird, bestimmt das Kommandozeilen-Flag `--theme` oder die Einstellung `theme:` in der `markpublish.yaml` (Standard: `default`).

Rang	Ort
1. Benutzer	`~/.markpublish/templates/<theme>/` bzw. benutzerspezifisches AppData-Verzeichnis
2. Projekt	`--templates-dir` auf der CLI, `templates_dir:` in der YAML, `MARKPUBLISH_TEMPLATES_DIR` oder `./templates/<theme>/`
3. Paket	Mitgeliefertes Standard-Theme `default`

`markpublish templates` zeigt, welche Vorlagen auf dem System vorhanden sind und woher sie stammen.

Anpassen eines Themes

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

Kopiert das eingebaute Theme ins Projekt, wo Rang 2 es beim nächsten Build automatisch aufgreift. Für PDF enthält das Theme:

- `template.typ`: Hauptlayout-Vorlage in Typst
- `i18n.yaml`: Übergreifende Beschriftungen des Themes
- `pdf/i18n.yaml`: Formatspezifische Beschriftungen für PDF
- `fonts/`: Optionale Schriftarten (.ttf, .otf)
- `assets/`: Grafiken, Logos und Icons

© 2026 Frank Winter

Statische Texte (i18n)

Labels lösen über eine vierstufige Kaskade auf; eine tiefere Ebene überschreibt nur die Schlüssel, die sie tatsächlich setzt:

Ebene	Datei
1. Programm	`markpublish/i18n.yaml` (Standardtexte der Anwendung)
2. Theme	`<templates>/<theme>/i18n.yaml`
3. Zielformat	`<templates>/<theme>/pdf/i18n.yaml`
4. Projekt	`./i18n.yaml` neben der `markpublish.yaml`

Jede Datei ist nach Sprachcode gegliedert (`de:`, `en:`); der Schlüssel `"*"` gilt für jede Sprache. Ebene 4 gehört dem Projekt: Sie erlaubt das Überschreiben von Texten und das Lokalisieren freier Metadatenfelder (`abteilung: "Abteilung"`). Mit `markpublish labels --overridden` bzw. `markpublish labels --theme <theme>` prüfen Sie die Kaskade vor dem Bauen.

Wichtige CLI-Befehle

Befehl	Wichtige Optionen	Bedeutung
`markpublish build`	`--theme`, `--target`, `-o`, `--templates-dir`	Dokument nach PDF kompilieren
`markpublish init`	`[ZIEL]`, `--title`, `--lang`	Minimales Starterprojekt einrichten
`markpublish cheatsheet`	`--lang`, `--theme`, `-o`, `--templates-dir`	Diese Schnellreferenz als PDF erzeugen
`markpublish manual`	`--lang`, `--theme`, `-o`, `--templates-dir`	Vollständiges Benutzerhandbuch erzeugen
`markpublish templates`	`--target`, `--templates-dir`	Vorlagen und Auflösungsrang auflisten
`markpublish export-template`	`[THEME]`, `[ZIEL]`, `--target`	Vorlagendateien zur Anpassung exportieren
`markpublish labels`	`--theme`, `--target`, `--templates-dir`, `--overridden`	Statische Texte und Metadaten analysieren

© 2026 Frank Winter