markpublish Benutzerhandbuch

Moderne PDF-Dokumentenerstellung aus Markdown

> *Praxisorientierte Anleitung und Referenz zur Konfiguration, CLI-Bedienung, Theme-Gestaltung und Dokumentenerstellung mit markpublish.*

Version	2.4.1
Datum	13.10.2026
Autor	Frank Winter
Copyright	© 2026 Frank Winter

Inhaltsverzeichnis

1 Einleitung **4**
1.1 Motivation und Zielsetzung 5
1.2 Kernfeatures 5
1.3 Architektur im Überblick 6
1.4 Nutzung von KI bei der Softwareentwicklung 6

2 Installation und Schnellstart **7**
2.1 Voraussetzungen 8
2.2 Installation 8
2.3 Schritte bis zum ersten Build 8

3 Kommandozeilen-Schnittstelle (CLI) **10**
3.1 Befehlsübersicht 11
3.2 markpublish build 11
3.3 markpublish init 13
3.4 markpublish manual / markpublish cheatsheet 14
3.5 markpublish templates 15
3.6 markpublish export-template 16
3.7 markpublish labels 16
3.8 Globale Optionen und Umgebungsvariablen 18

4 Projektkonfiguration mit markpublish.yaml **20**
4.1 Die drei Strukturebenen 21
4.2 Eigene Metadatenfelder 23
4.3 Platzhalter im Text 24
4.4 Werte aus anderen Dateien (`!file`) 24
4.5 Beispiel einer vollständigen Konfiguration 25

5 Templates und Mehrsprachigkeit **27**
5.1 Die Auflösungshierarchie für Themes 28
5.2 Aufbau eines Themes 28
5.3 Eigene Themes erstellen und anpassen 30
5.4 Mehrsprachigkeit und Textkaskade (i18n) 31

© 2026 Frank Winter

6 Besondere Markdown-Features ... **33**
6.1 Hinweisboxen (Admonitions und Callouts) ... 34
6.2 Definitionslisten ... 36
6.3 Aufgabenlisten (Tasklists) ... 37
6.4 Fußnoten ... 37
6.5 Mathematische Formeln ... 38
6.6 Typografische Textauszeichnungen und Satzzeichen ... 39
6.7 Automatische Symbole und Pfeile ... 39
6.8 Bilder, Abbildungen und Tabellen ... 40
6.9 Automatische Verlinkung und Querverweise ... 42
6.10 Maskierungsfreie Sonderzeichen ... 42

7 Daten im Text ... **44**
7.1 Platzhalter ... 45
7.2 Schleifen ... 45
7.3 Kurznamen ... 46
7.4 Nachschlagen ... 46
7.5 Bedingungen ... 47
7.6 Regeln für Anweisungen ... 47
7.7 Beispiel: Datenflüsse und Systeme ... 47

Anhänge **49**

Anhang A: Schema-Referenz markpublish.yaml ... **50**
A.1 Dokument-Ebene (document) ... 50
A.2 Globale Projekt-Einstellungen ... 52
A.3 Abschnitts-Ebene (parts) ... 53
A.4 Kapitel-Ebene (chapters) ... 54
A.5 Verzeichnis-Steuerung (Scope-Werte) ... 56

Anhang B: Fehlerbehebung und FAQ ... **57**
B.1 Diagnose von Kompilierungsfehlern ... 57
B.2 Häufige Ursachen und Lösungen ... 57
B.3 Häufig gestellte Fragen (FAQ) ... 60

© 2026 Frank Winter

KAPITEL 1

Einleitung

> Motivation, Kernfeatures und die Architektur von markpublish.

AUF EINEN BLICK

1.1 Motivation und Zielsetzung 5

1.2 Kernfeatures 5

1.3 Architektur im Überblick 6

1.4 Nutzung von KI bei der Softwareentwicklung 6

1.1 Motivation und Zielsetzung

Markdown ist das führende Format für technische Dokumentationen, Notizen und Projektberichte. Seine Stärke liegt in der Einfachheit und Lesbarkeit im reinen Textformat. Sobald jedoch aus diesen Texten hochwertige, druckfertige Berichte, Handbücher oder Whitepapers entstehen sollen, stoßen herkömmliche Werkzeuge an ihre Grenzen. Häufig folgt ein zeitaufwendiger manueller Nachbearbeitungsschritt in Textverarbeitungs- oder Layoutprogrammen.

markpublish schließt diese Lücke: Es verwandelt strukturierte Markdown-Dateien über eine einfache, aber mächtige Konfiguration vollautomatisch in typografisch anspruchsvolle, professionell gestaltete PDF-Dokumente.

Das Ziel von markpublish ist es, Entwicklern, technischen Redakteuren und Autoren einen nahtlosen Veröffentlichungsprozess zu bieten. Inhalte werden in einfachem Markdown verfasst, während markpublish die vollständige Kontrolle über Layout, Gliederung, Trennseiten, Nummerierung, Verzeichnisse und Typografie übernimmt.

1.2 Kernfeatures

markpublish zeichnet sich durch einen klaren Fokus auf Dokumentenqualität, Geschwindigkeit und Verlässlichkeit aus:

- **Moderne Satzqualität durch Typst:** Die Satz-Engine Typst sorgt für typografische Präzision, perfekten Randausgleich und ein ästhetisches Layout.
- **Hohe Kompiliergeschwindigkeit:** Selbst umfangreiche Dokumente mit Dutzenden von Seiten, Abbildungen und Tabellen werden in rund ein bis zwei Sekunden vollständig gesetzt.
- **Einfache Installation und Portabilität:** Als reguläres Python-Paket lässt sich markpublish plattformübergreifend auf Windows, macOS und Linux ohne komplizierte Einrichtung betreiben.
- **Deklarative Konfiguration:** Eine einzige Datei (`markpublish.yaml`) steuert das gesamte Projekt.
- **Mehrstufige Dokumentenarchitektur:** Unterstützung einer klaren Zweiteilung in übergeordnete Abschnitte (*Parts*) und inhaltstragende Kapitel (*Chapters*) mit flexibler Trennseiten-Steuerung.
- **Präzise Verzeichnisse und Nummerierung:** Vollautomatische Generierung von Gesamtinhaltsverzeichnis, Abschnittsverzeichnissen und lokalen Kapitelverzeichnissen sowie flexibel konfigurierbare Kapitel- und Überschriftennummerierung.
- **Umfassende Markdown-Erweiterungen:** Standardmäßige Unterstützung von GitHub-konformen Hinweisboxen (Admonitions/Callouts), Tabellen, Definitionslisten, Fußnoten, Aufgabenlisten und Quellcode mit Syntax-Highlighting.

© 2026 Frank Winter

- **Kaskadierende Mehrsprachigkeit (i18n):** Mehrsprachige Dokumente und Themes (statische Texte wie Inhaltsverzeichnis, Kapitel, Seitenzahlen) über eine vierstufige Kaskade. Auch die Programmoberfläche (CLI) ist zweisprachig (Deutsch und Englisch) und folgt der Systemsprache.

1.3 Architektur im Überblick

Die Arbeitsweise von markpublish folgt einer klaren Verarbeitungs-Pipeline:

1. **Konfigurations- und Dokumentenanalyse:** markpublish liest die Datei `markpublish.yaml` ein, validiert sämtliche Einstellungen und baut die Gliederungshierarchie aus Abschnitten und Kapiteln auf.
2. **Inhaltsaufbereitung:** Die Markdown-Dateien der Kapitel werden über eine Syntaxbaum-Pipeline (ElementTree AST) eingelesen und durch einen nativen Typst-Serializer (`TypstSerializer`) direkt in sauberes Typst-Markup überführt. Überschriften und Verzeichnisse werden auf AST-Ebene synchronisiert, Sprungmarken gesetzt und lokale Bildpfade automatisch für den Bau isoliert.
3. **Template-Zusammenstellung:** Das gewählte Theme (standardmäßig das integrierte Standard-Theme) stellt die Layout-Vorgaben bereit. Metadaten (gesammelt in einem typisierten `meta`-Wörterbuch), Kopf- und Fußzeilen, Trennseiten und Textinhalte werden präzise in die Typst-Umgebung übergeben.
4. **Kompilierung zum PDF:** Die Typst-Engine kompiliert das Gesamtdokument in einem einzigen, hocheffizienten Durchlauf direkt in die fertige PDF-Datei. Tritt ein Fehler auf, wird der generierte Typst-Code zur schnellen Diagnose in `.markpublish/last_failed_build.typ` abgelegt.

1.4 Nutzung von KI bei der Softwareentwicklung

Bei der Entwicklung von markpublish wurden verschiedene KI-Modelle bewusst als Werkzeuge eingesetzt: beim Entwerfen und Schreiben von Code ebenso wie als Gesprächspartner für Architekturentscheidungen, für Anregungen zu Bedienung und Typografie und für gegenseitige Code-Reviews, bei denen ein Modell die Entwürfe eines anderen prüft. Das hat die Entwicklung beschleunigt und die Codequalität spürbar verbessert.

Die Grenzen waren ebenso deutlich: Fachlicher Kontext, die Richtung des Projekts und der letzte Feinschliff kommen nicht aus einem Sprachmodell. Jede Änderung wurde vom Autor geprüft, zusammengeführt und verantwortet.

markpublish selbst enthält keine KI-Funktionen. Alle Dokumente werden ausschließlich lokal auf Ihrem Rechner verarbeitet.

© 2026 Frank Winter

KAPITEL 2

Installation und Schnellstart

> Der direkte Weg von der Installation bis zum ersten fertigen PDF.

AUF EINEN BLICK

2.1 Voraussetzungen 8

2.2 Installation 8

2.3 Schritte bis zum ersten Build 8

Dieses Kapitel beschreibt den direkten Weg von der Installation bis zum ersten fertig erstellten PDF-Dokument.

2.1 Voraussetzungen

Für den Betrieb von markpublish wird eine funktionierende Python-Installation benötigt:

- **Python-Version:** 3.10 oder neuer
- **Betriebssystem:** Windows, macOS oder Linux

2.2 Installation

markpublish wird komfortabel über den Python-Paketmanager `pip` installiert:

```
pip install markpublish
```

Nach Abschluss der Installation steht der Befehl `markpublish` (sowie das kurze Alias `mpub`) im Terminal zur Verfügung. Die erfolgreiche Bereitstellung lässt sich mit folgendem Aufruf prüfen:

```
markpublish --version
```

2.3 Schritte bis zum ersten Build

Die Erstellung eines neuen Dokumentenprojekts erfolgt in wenigen einfachen Schritten:

2.3.1 Neues Projekt initialisieren

Mit dem Befehl `init` legt markpublish ein neues Projektverzeichnis mit einer minimalen, lauffähigen Startstruktur an:

```
markpublish init mein-dokument
```

Besteht das angegebene Verzeichnis bereits und ist nicht leer, bricht `init` vorsorglich ab, um ein versehentliches Überschreiben vorhandener Dateien zu verhindern.

Dieser Befehl erzeugt einen neuen Ordner `mein-dokument/` mit folgender Grundstruktur:

© 2026 Frank Winter

```
mein-dokument/
├── markpublish.yaml
├── kapitel_1.md
└── kapitel_2.md
```

2.3.2 Erstes PDF erstellen

Sie können das PDF direkt aus dem übergeordneten Ordner über den Pfad zur Konfiguration erzeugen (diese Zeile gibt `init` am Ende auch als Empfehlung aus):

```
markpublish build mein-dokument/markpublish.yaml
```

Oder Sie wechseln zuerst in das neu angelegte Verzeichnis und starten den Build dort:

```
cd mein-dokument
markpublish build
```

markpublish liest die Konfigurationsdatei `markpublish.yaml`, verarbeitet die referenzierten Markdown-Dateien und kompiliert das Dokument.

2.3.3 Dokument öffnen

Das generierte PDF befindet sich nun direkt in Ihrem Projektverzeichnis:

```
mein-dokument/
└── mein_dokument.pdf
```

(Hinweis: Der Dateiname der Ausgabe leitet sich automatisch aus dem in der `markpublish.yaml` hinterlegten Dokumententitel ab – Leerzeichen und Bindestriche werden dabei zu Unterstrichen. Ohne `--title` trägt das neue Dokument den Namen des Projektordners, hier also `mein-dokument` → `mein_dokument.pdf`.)

Öffnen Sie die Datei mit einem beliebigen PDF-Betrachter, um das Ergebnis zu begutachten. Das Dokument enthält die formatierte Einstiegsseite mit Kopf- und Fußzeilen.

© 2026 Frank Winter

KAPITEL 3

Kommandozeilen-Schnittstelle (CLI)

Detaillierte Befehlsübersicht und vollständige Optionen-Referenz aller Befehle.

AUF EINEN BLICK

3.1 Befehlsübersicht 11

3.2 markpublish build 11

3.3 markpublish init 13

3.4 markpublish manual / markpublish cheatsheet 14

3.5 markpublish templates 15

3.6 markpublish export-template 16

3.7 markpublish labels 16

3.8 Globale Optionen und Umgebungsvariablen 18

Die Kommandozeile ist die primäre Schnittstelle zur Arbeit mit markpublish. Neben dem Hauptbefehl `markpublish` steht das kurze Alias `mpub` zur Verfügung.

3.1 Befehlsübersicht

Befehl	Kurzbeschreibung
`build`	Kompiliert das Dokument auf Basis der Konfigurationsdatei.
`init`	Erstellt ein neues Projekt mit Konfigurationsdatei und Beispielkapitel.
`manual`	Rendert das offizielle Benutzerhandbuch aus dem installierten Paket.
`cheatsheet`	Rendert die kompakte zweiseitige Kurzreferenz aus dem Paket.
`templates`	Listet alle verfügbaren Templates und deren Herkunftsebenen auf.
`export-template`	Exportiert das Standard-Theme zur individuellen Anpassung in das Projekt.
`labels`	Zeigt alle aufgelösten Textbeschriftungen und deren Herkunftsebene (i18n) an.

3.2 markpublish build

Der Befehl `build` führt den Veröffentlichungsprozess aus. Er liest die Konfiguration ein, verarbeitet alle Markdown-Dateien der Kapitel und kompiliert das fertige Dokument.

```
markpublish build [CONFIG_FILE] [OPTIONEN]
```

3.2.1 Optionen

Option	Default	Bedeutung
`CONFIG_FILE` *(Argument)*	`markpublish.yaml`	Pfad zur Projekt-Konfigurationsdatei.
`--target`, `-t`	`pdf`	Ausgabeformat (`pdf`).
`--output`, `-o`	*(automatisch)*	Pfad zur Zieldatei oder zum Zielverzeichnis. Endet der Pfad ohne Dateiendung, wird er als Verzeichnis behandelt.

© 2026 Frank Winter

Option	Default	Bedeutung
`--theme`	`None`	Rendert mit einem alternativen Theme anstelle der Angabe in der Konfigurationsdatei.
`--templates-dir`	`None`	Pfad zu einem benutzerdefinierten Vorlagen-Verzeichnis.

3.2.2 Erläuterungen

Ohne Argument sucht markpublish die Datei `markpublish.yaml` im aktuellen Verzeichnis – im Projektordner genügt also der blanke Aufruf `markpublish build`. Wird ein anderer Pfad angegeben, gilt dessen Verzeichnis als Projektordner: Sämtliche relativen Angaben der Konfiguration (Kapiteldateien, Bilder, Vorlagen) werden von dort aus aufgelöst, nicht vom aktuellen Arbeitsverzeichnis.

Wird kein `--output` angegeben, erzeugt markpublish die Datei im Projektordner. Der Dateiname leitet sich automatisch aus dem Dokumententitel ab (z. B. `mein_dokument.pdf`).

`--templates-dir` erwartet das Sammelverzeichnis, nicht das Theme selbst. Welches Theme verwendet wird, bestimmt entweder die Option `--theme` auf der Kommandozeile oder `theme:` in der `markpublish.yaml` (Standard: `default`). markpublish setzt beide Angaben zusammen und sucht unter `<verzeichnis>/<theme>/<zielformat>`:

```
meine-vorlagen/          <- hierhin zeigt --templates-dir
└── firma/               <- diesen Namen trägt theme: in der markpublish.yaml
    └── pdf/             <- das Zielformat
        ├── template.typ
        └── i18n.yaml
```

Der zugehörige Aufruf lautet damit:

```
markpublish build --templates-dir meine-vorlagen
```

Ein relativer Pfad wird vom Projektordner aus gelesen. Dasselbe Verzeichnis lässt sich auch dauerhaft hinterlegen; es gilt die erste Angabe, die vorhanden ist:

1. `--templates-dir` auf der Kommandozeile
2. `templates_dir:` in der `markpublish.yaml`
3. die Umgebungsvariable `MARKPUBLISH_TEMPLATES_DIR`
4. ein Ordner `templates/` neben der Konfigurationsdatei

© 2026 Frank Winter

Das so bestimmte Verzeichnis ist nur eine der drei Ebenen, die markpublish durchsucht; eine gleichnamige Vorlage im Home-Verzeichnis hat weiterhin Vorrang (siehe Kapitel *Templates und Mehrsprachigkeit*).

Beispiele:

```
# Standard-Build im aktuellen Projekt
markpublish build

# Mit einem alternativen Theme bauen (übersteuert die Angabe in
markpublish.yaml)
markpublish build --theme custom-theme

# Andere Konfigurationsdatei und anderen Ausgabeordner festlegen
markpublish build projekte/bericht.yaml --output dist/
```

3.3 markpublish init

Der Befehl `init` dient dem schnellen Aufsetzen eines neuen Projekts. Er erzeugt das Verzeichnis (sofern noch nicht vorhanden), eine lauffähige `markpublish.yaml` sowie ein erstes Markdown-Kapitel.

```
markpublish init [ZIELORDNER] [OPTIONEN]
```

3.3.1 Optionen

Option	Default	Bedeutung
`ZIELORDNER` *(Argument)*	`.` *(aktueller Ordner)*	Zielverzeichnis, in dem das Projekt initialisiert werden soll.
`--title`, `-t`	*(Name des Zielordners)*	Titel des neuen Dokuments in der erzeugten Konfiguration.
`--lang`, `-l`	*(Systemsprache)*	Sprache der Vorlage (`de`, `en`).

3.3.2 Erläuterungen

Die erzeugte `markpublish.yaml` und das Einstiegskapitel stammen aus den mitgelieferten Vorlagen. Mit `--lang` (z. B. `--lang de`) lässt sich die Sprache der Vorlage explizit vorgeben; ohne

© 2026 Frank Winter

Angabe wird automatisch die Systemsprache ermittelt und eingetragen, sodass das Projekt auf jedem Rechner konsistent kompiliert.

Besteht das Zielverzeichnis bereits und enthält Dateien, bricht `init` mit einer Fehlermeldung ab. Ein bestehendes Verzeichnis wird niemals stillschweigend überschrieben; für eine Neuinitialisierung wählen Sie entweder einen neuen Ordner oder stellen sicher, dass das bestehende Verzeichnis leer ist.

Beispiel:

```
markpublish init mein-handbuch --lang de --title "Benutzerhandbuch System X"
```

3.4 markpublish manual / markpublish cheatsheet

markpublish bringt seine vollständige Dokumentation direkt im Paket mit. Beide Befehle erzeugen die Dokumentation on-demand aus den installierten Quellen – das Ergebnis beschreibt somit immer exakt die aktuell installierte Version.

- `markpublish manual`: Rendert dieses umfassende Benutzerhandbuch.
- `markpublish cheatsheet`: Rendert eine kompakte Schnellreferenz.

```
markpublish manual [OPTIONEN]
markpublish cheatsheet [OPTIONEN]
```

3.4.1 Optionen

Option	Default	Bedeutung
`--lang`, `-l`	*(Systemsprache)*	Sprachauswahl des Dokuments (z. B. `de` oder `en`).
`--target`, `-t`	`pdf`	Ausgabeformat (`pdf`).
`--output`, `-o`	`.` *(aktueller Ordner)*	Zielpfad für das generierte Dokument.
`--theme`	`None`	Rendert mit einem alternativen Theme anstelle des Standard-Themes.
`--templates-dir`	`None`	Pfad zu einem benutzerdefinierten Vorlagen-Verzeichnis.

© 2026 Frank Winter

3.4.2 Erläuterungen

Wird beim Aufruf von `manual` oder `cheatsheet` keine Sprache mit `--lang` angegeben, ermittelt markpublish automatisch die Sprache Ihres Betriebssystems. Ist für diese Sprache keine Übersetzung vorhanden, greift markpublish auf die Standardsprache des Dokuments zurück.

Beispiel:

```
# Deutsches Benutzerhandbuch im aktuellen Ordner erzeugen
markpublish manual --lang de

# Englische Kurzreferenz erzeugen
markpublish cheatsheet --lang en
```

3.5 markpublish templates

Listet alle im System und Projekt verfügbaren Vorlagen auf und zeigt übersichtlich an, welche Vorlage im Falle von Namensgleichheiten Vorrang hat.

```
markpublish templates [OPTIONEN]
```

3.5.1 Optionen

Option	Default	Bedeutung
`--target`, `-t`	`None`	Filtert die Anzeige nach Zielformat (`pdf`).
`--templates-dir`	`None`	Zusätzliches Vorlagen-Verzeichnis, das in die Suche einbezogen werden soll.

3.5.2 Erläuterungen

Die Ausgabe erfolgt als tabellarische Übersicht mit Angabe von Theme-Name, Herkunftsebene (User, Common oder Package) und einem Status-Indikator, welches Theme bei einem Build aktiv verwendet wird.

© 2026 Frank Winter

3.6 markpublish export-template

Exportiert ein mitgeliefertes Theme aus dem markpublish-Paket in ein lokales Verzeichnis, damit es nach eigenen Wünschen angepasst oder als Vorlage für ein eigenes Corporate Design verwendet werden kann.

```
markpublish export-template [THEME] [ZIELVERZEICHNIS] [OPTIONEN]
```

3.6.1 Optionen

Option	Default	Bedeutung
`THEME` *(Argument)*	`default`	Name des zu exportierenden Themes.
`ZIELVERZEICHNIS` *(Argument)*	`templates`	Ordner, in den die Vorlagendateien kopiert werden sollen.
`--target`, `-t`	`all`	Zielformat der Vorlagen (`pdf` oder `all`).

3.6.2 Erläuterungen

Beim Export werden die Typst-Vorlagendateien (`template.typ`) sowie die zugehörige Beschriftungsdatei `i18n.yaml` in das Zielverzeichnis kopiert. Liegt ein Theme im Projektordner unter `templates/`, greift markpublish automatisch darauf zu.

Beispiel:

```
# Exportiert das Standard-Theme in den lokalen Ordner ./templates
markpublish export-template default ./templates
```

3.7 markpublish labels

Zeigt eine detaillierte Aufstellung aller statischen Beschriftungen (wie „Inhaltsverzeichnis“, „Kapitel“, „Seite X von Y“) für die im Projekt gewählte Sprache sowie deren Herkunftsebene in der Kaskade.

```
markpublish labels [CONFIG_FILE] [OPTIONEN]
```

© 2026 Frank Winter

3.7.1 Optionen

Option	Default	Bedeutung
`CONFIG_FILE` *(Argument)*	`markpublish.yaml`	Pfad zur Projekt-Konfigurationsdatei.
`--target`, `-t`	`pdf`	Zielformat, dessen Beschriftungskaskade angezeigt werden soll.
`--theme`	`None`	Prüft die Beschriftungskaskade für ein alternatives Theme.
`--templates-dir`	`None`	Benutzerdefiniertes Vorlagen-Verzeichnis.
`--overridden`	`False`	Zeigt ausschließlich Beschriftungen an, die durch ein Theme oder Projekt überschrieben wurden.

3.7.2 Erläuterungen

Dieser Befehl ist das zentrale Diagnosewerkzeug bei der Theme-Entwicklung und Dokumentenkonfiguration. Er prüft noch vor dem eigentlichen Build, ob alle Metadatenfelder, Beschriftungen und Theme-Signaturen vollständig zueinander passen.

Die tabellarische Ausgabe gliedert sich in acht Spalten:

Spalte	Bedeutung
Schlüssel	Name des Metadatums oder Textschlüssels (z. B. `title`, `author`, `toc_title` oder eigene Felder wie `abteilung`).
Theme-Nutzung	Art der Verwendung im Theme: `key` (als Beschriftungsbezeichner), `wert` (als Inhaltswert), `key/wert` (beides) oder `-` (vom Theme nicht verwendet).
Label	Aufgelöster Text aus der Beschriftungskaskade für diesen Schlüssel. Fehlt ein benötigtes Label, wird dies rot markiert.
i18n-Quelle	Ebene der Kaskade, aus der das Label stammt (`mpub`, `theme`, `target`, `projekt`). Grün hervorgehoben sind Ebenen, die den Standard überschrieben haben.
Label-Fallback	Im Theme definierter Notfall-Ersatztext, falls das Label in keiner i18n-Ebene existiert.
Wert	Im Dokument (`document:`) hinterlegter konkreter Wert.
Wert-Fallback	Im Theme notierter Fallback-Wert für den Fall, dass das Feld im Dokument nicht gesetzt ist.

© 2026 Frank Winter

Spalte	Bedeutung
Status	Detaillierter Befund: `OK`, `ungenutzt`, `Label fehlt`, `Wert nicht gesetzt` oder rot hervorgehobene kritische Fehler, die einen Build-Abbruch zur Folge hätten.

Die Ebenen in der Spalte **i18n-Quelle** bedeuten:

Wert	Datei
`mpub`	Der Programmstandard, `markpublish/i18n.yaml`
`theme`	`<theme>/i18n.yaml`
`target`	`<theme>/<zielformat>/i18n.yaml`
`projekt`	`i18n.yaml` neben Ihrer `markpublish.yaml`

Grün hervorgehoben sind Ebenen, die eigene Werte gegenüber dem Programmstandard definieren – auf diese beschränkt `--overridden` die Ausgabe. Zeilen, die den Build zum Abbruch bringen würden, bleiben dabei immer sichtbar: ein Filter, der einen Fehler verschweigt, wäre eine Falle.

Der Sprachblock steht nur dann in Klammern dabei, wenn er von der Dokumentsprache abweicht: `(*)` für einen sprachunabhängigen Eintrag, `(en)` für einen Rückfall auf die Fallback-Sprache. Am Fuß der Ausgabe stehen die vollständigen Pfade zu allen vier Ebenen, jeweils mit dem Vermerk, ob die Datei dort vorhanden ist.

Vorab-Signaturprüfung und Bilanz:
Zusätzlich prüft `markpublish labels` die Typst-Vorlage des Themes direkt gegen den Aufruf, den markpublish beim Rendern erzeugt. Deklariert das Theme notwendige Parameter nicht (wie `meta: (:)`), meldet `labels` den Signaturfehler sofort vor dem Bauen. Eine zusammenfassende Bilanz am Ende der Ausgabe zählt kritische Abbrüche, Warnungen sowie ungenutzte Felder übersichtlich auf.

3.8 Globale Optionen und Umgebungsvariablen

Folgende Optionen und Umgebungsvariablen steuern das Verhalten von markpublish übergeordnet:

Option / Variable	Bedeutung
`--version`, `-v`	Gibt die installierte Version von markpublish aus und beendet das Programm.

© 2026 Frank Winter

Option / Variable	Bedeutung
`--help`	Zeigt die integrierte Hilfe und Parameterübersicht im Terminal an.
`--ui-lang`	Sprache der Bildschirmausgaben und Terminalmeldungen für diesen Aufruf (`de`, `en`).
`MARKPUBLISH_UI_LANG`	Umgebungsvariable zur dauerhaften Festlegung der Terminal-Sprache.
`MARKPUBLISH_TEMPLATES_DIR`	Umgebungsvariable für ein alternatives globales Vorlagenverzeichnis.

Die Anzeigesprache der Programmoberfläche ermittelt markpublish in folgender Prioritätsreihenfolge:

1. Das Kommandozeilen-Flag `--ui-lang`
2. Die Umgebungsvariable `MARKPUBLISH_UI_LANG`
3. Die Systemsprache des Betriebssystems (Windows Anzeigesprache bzw. POSIX-Variablen `LC_ALL`, `LANG`)
4. Englisch (`en`) als Ausweichsprache

(Hinweis: Die Benutzeroberflächensprache betrifft ausschließlich die Terminalausgaben und Hilfetexte des Programms. Die Sprache des generierten PDF-Dokuments und dessen Silbentrennung wird davon völlig unabhängig über `language:` in der `markpublish.yaml` festgelegt.)

```
# Gibt Meldungen für diesen Aufruf auf Englisch aus
markpublish --ui-lang en build
```

© 2026 Frank Winter

KAPITEL 4

Projektkonfiguration mit markpublish.yaml

Strukturierung in Document, Parts und Chapters anhand praktischer Beispiele.

AUF EINEN BLICK

4.1 Die drei Strukturebenen 21

4.2 Eigene Metadatenfelder 23

4.3 Platzhalter im Text 24

4.4 Werte aus anderen Dateien (!file) 24

4.5 Beispiel einer vollständigen Konfiguration 25

Die zentrale Steuerungsdatei eines jeden markpublish-Projekts ist die Datei `markpublish.yaml`. Sie definiert das Dokument deklarativ: Metadaten, Layout-Vorgaben, Gliederung und die Zuordnung der Markdown-Dateien zu Kapiteln und Abschnitten.

4.1 Die drei Strukturebenen

markpublish organisiert Dokumente in drei klar voneinander abgegrenzten Ebenen:

```
document
└── parts (Abschnitte)
    └── chapters (Kapitel)
```

Tiefer reicht die Gliederung der Konfiguration nicht. Innerhalb eines Kapitels strukturiert die Markdown-Datei selbst weiter, mit `##`, `###` und so fort.

4.1.1 Document (Dokumentebene)

Die oberste Ebene `document:` beschreibt das Gesamtdokument:

Metadaten Titel, Untertitel, Autoren, Version, Datum, Sprache, Status und Copyright-Hinweise.

Layout-Schalter Anzeige des Deckblatts (`cover`), Aktivierung von Kopfzeilen (`header`) und Fußzeilen (`footer`).

Globale Verzeichnisvorgaben Steuerung der Tiefe für das Gesamt-Inhaltsverzeichnis (`document_toc`), Abschnittsverzeichnisse (`part_toc`) und lokale Kapitelverzeichnisse (`chapter_toc`).

> **Keine Wahrheitswerte bei Verzeichnisschaltern**
> Die Schalter `document_toc`, `part_toc` und `chapter_toc` akzeptieren **keine** Booleans (`true` oder `false`). Verwenden Sie `"full"` (volle Tiefe), `"none"` (kein Verzeichnis) oder eine positive Zahl ab `1` für die maximale Gliederungstiefe (z. B. `2`). Ein notiertes `document_toc: false` bricht die Validierung mit einer deutlichen Meldung ab.

© 2026 Frank Winter

Dokumente aus einem einzigen Kapitel

Besteht das Dokument aus genau einem Kapitel, lässt das Gesamt-Inhaltsverzeichnis dessen eigene Zeile weg. Sie wiederholte nur den Titel des Deckblatts, und ein Eintrag ohne Geschwister unterscheidet nichts von etwas anderem. Die Unterüberschriften des Kapitels bleiben aufgeführt — anders als bei `document_toc: "none"`, das den gesamten Ast aus dem Verzeichnis nimmt. Verloren geht dabei nichts: die Kapitelnummer steht weiterhin über dem Text beziehungsweise als Marke auf der Trennseite, und das Verzeichnis eines Abschnitts führt sein Kapitel unverändert auf. Sobald ein zweites Kapitel hinzukommt, erscheinen beide Zeilen.

Die **Nummerierung** bleibt davon unberührt: die Abschnitte tragen weiterhin die Kapitelnummer im Präfix (`1.1`, `1.2`). Soll auch sie die Kapitelebene überspringen, notieren Sie unter `document:` das Muster `autonum_pattern: "_|_|1|.1|+"` — Part und Kapitel ohne Nummer, die H2 wird zur 1., 2., 3., die H3 zu 1.1. Kommt später ein zweites Kapitel hinzu, gehört diese Zeile wieder entfernt, sonst zählen die H2 über die Kapitelgrenze hinweg durch.

Die Tiefenangabe `document_toc` zählt dabei **Gliederungsebenen, nicht sichtbare Zeilen**. Die Kapitelebene ist übersprungen, aber nicht verschwunden: eine H3 liegt auf Ebene 3, gleich ob die Kapitelzeile gedruckt wird. Für zwei sichtbare Ebenen (H2 und H3) notieren Sie deshalb `document_toc: 3`. Das ist so gewollt — andernfalls bedeutete dieselbe Zahl je nach Anzahl der Kapitel etwas anderes.

Nummerierungsregeln Vorgaben zur automatischen Nummerierung (`autonum_pattern`, `autonum_reset`).

4.1.2 Parts (Abschnitte)

Ein Dokument wird durch `parts:` in übergeordnete Abschnitte gegliedert (z. B. „Hauptteil“, „Fallstudien“, „Anhänge“):

- Ein Dokument muss immer über `parts:` aufgebaut sein; ein flaches `chapters:` direkt auf oberster Ebene wird abgewiesen.
- Ein Abschnitt fasst logisch zusammengehörige Kapitel zusammen.
- Ohne Angabe tritt ein Abschnitt selbst nicht in Erscheinung (`break_before: "none"`): er klammert seine Kapitel und vererbt seine Einstellungen, belegt aber keine Seite und erscheint nicht im Inhaltsverzeichnis. Mit `break_before: "divider"` erhält er eine gestaltete Trennseite, mit `break_before: "page"` eine Überschrift auf einer neuen Seite.
- Auf Abschnittsebene gesetzte Einstellungen (z. B. Verzeichnistiefe oder ein `autonum_pattern`) vererben sich automatisch auf alle darin enthaltenen Kapitel.
- Auf `parts:` sind ausschließlich die deklarierten Konfigurationsschlüssel erlaubt. Unbekannte Schlüssel (wie `break_befor`) werden mit Ähnlichkeitsvorschlägen abgelehnt, um unbemerkte Fehlkonfigurationen auszuschließen.

© 2026 Frank Winter

4.1.3 Chapters (Kapitel)

Die Liste `chapters:` innerhalb eines Abschnitts verweist auf die eigentlichen Markdown-Inhaltsdateien:

- Jedes Kapitel verweist auf seine Markdown-Datei (`file:`). Autoren können in ihren Markdown-Dateien ganz natürlich mit einer `#`-Überschrift beginnen.
- Über **`show_title:`** (`true` oder `false`, Standard: `true`) lässt sich steuern, ob die `#`-Überschrift der Markdown-Datei auf der Inhaltsseite gerendert werden soll. Wird vor dem Kapitel eine Trennseite erzeugt (`break_before: "divider"`), kann mit `show_title: false` verhindert werden, dass der Titel auf der Folgeseite doppelt erscheint – die Inhaltsseite beginnt dann direkt mit dem Fließtext oder der ersten Zwischenüberschrift.
- Über die optionalen Schlüssel **`toc_title:`** (für Inhaltsverzeichnisse und Kopfzeilen) und **`divider_title:`** (für Trennseiten) können gezielt abweichende Titel vergeben werden (z. B. eine prägnante Kurzform im Inhaltsverzeichnis gegenüber einer ausführlichen Überschrift auf der Textseite). Fehlen die Schlüssel, erben beide automatisch die `#`-Überschrift der Datei.
- Soll eine Trennseite erzeugt werden (`break_before: "divider"`) oder das Kapitel im Inhaltsverzeichnis gelistet werden, muss mindestens eine `#`-Überschrift oder der entsprechende Titelschlüssel (`divider_title` / `toc_title`) vorhanden sein, andernfalls bricht der Build mit einer klaren Fehlermeldung ab.
- Kapitel können über das Feld `break_before` steuern, ob vor ihnen eine Trennseite erzeugt wird, ein einfacher Seitenwechsel erfolgt oder der Text nahtlos anschließt.

4.2 Eigene Metadatenfelder

Unter `document:` sind über die bekannten Schlüssel hinaus **beliebige eigene Felder** erlaubt. Sie erreichen das Theme, werden aber nicht von selbst gedruckt:

```yaml
document:
  title: "Projektbericht"
  abteilung: "Controlling"
  kunde: "Musterunternehmen GmbH"
  freigegeben: true
```

Das Metadatenraster des Deckblatts zeigt ausschließlich die Angaben, die das verwendete Theme dafür vorsieht – im mitgelieferten Standard-Theme sind das Version, Datum, Autor, Copyright und Status. Ein eigenes Feld kommt erst dazu, wenn das Theme es ausdrücklich aufführt (siehe Kapitel *Templates und Mehrsprachigkeit*).

© 2026 Frank Winter

> **Warum nicht automatisch?**
> Würde jedes freie Feld ungefragt auf dem Deckblatt landen, stünde dort auch jeder Tippfehler: Aus `abteilnug: "Controlling"` würde eine zusätzliche Zeile, die niemand angeordnet hat. So bleibt das Titelblatt die Entscheidung des Themes.

Drei Dinge sind dabei zu wissen:

- **Ungenutzte Felder meldet `markpublish labels`.** Ein Feld, das kein Theme abholt, erscheint dort mit dem Befund *ungenutzt* und am Fuß der Ausgabe noch einmal gesammelt – genau hier fällt auch ein vertippter Schlüssel auf. Ein Blick dorthin vor dem ersten Bauen erspart die Suche.
- **Die Beschriftung kommt aus der i18n-Kaskade.** Sobald ein Theme das Feld druckt, braucht es dazu einen Text; findet sich keiner, druckt das Deckblatt den Schlüssel selbst – also `abteilung` statt `Abteilung`. Legen Sie dafür eine `i18n.yaml` neben Ihre `markpublish.yaml`:

```yaml
# i18n.yaml, im Projektverzeichnis
de:
  abteilung: "Abteilung"
  kunde: "Kunde"
  freigegeben: "Freigegeben"
```

Diese Datei ist die letzte Stufe der Kaskade und gewinnt damit gegen Programm und Theme (siehe Kapitel *Templates und Mehrsprachigkeit*).

- **Der Typ bleibt erhalten.** Ein `true` ist ein Wahrheitswert und kein Text: Das Deckblatt setzt `Ja` oder `Nein` (beziehungsweise `Yes` / `No`), je nach Dokumentsprache. Die beiden Wörter stehen als `bool_true` und `bool_false` in der i18n-Kaskade.

4.3 Platzhalter im Text

Im Markdown setzt `{{author}}` eine Angabe aus `document:` ein, `{{custom.verfahren.name}}` einen eigenen Wert unter `custom:`. Dort dürfen Werte verschachtelt sein; sie erreichen das Theme nicht. Schleifen, Kurznamen, Nachschlagen und Bedingungen beschreibt Kapitel 7 „Daten im Text“.

4.4 Werte aus anderen Dateien (`!file`)

Statt einen Wert direkt hinzuschreiben, lädt `!file` ihn aus einer Datei oder von einer URL nach. Beides lässt sich beliebig mischen:

© 2026 Frank Winter

```yaml
document:
  custom:
    a: "xxx"
    verfahren: !file "../gemeinsam/verfahren.yaml"
    zweck: !file "zweck.txt"
    betreiber: !file "https://intranet.example/betreiber.yaml"
```

- **YAML-Dateien** (`.yaml`, `.yml`) setzen ihren Inhalt an die Stelle des Verweises, `{{custom.verfahren.name}}` greift also in die nachgeladene Datei. **Jede andere Datei** liefert ihren Text als einfachen Wert.
- **Relative Pfade** gelten relativ zu der Datei, in der das `!file` steht, bei einer URL relativ zu dieser URL.
- **Verschachtelt:** Eine nachgeladene YAML-Datei darf selbst wieder `!file` enthalten. Dieselbe Datei an mehreren Stellen ist erlaubt; ein Verweis, der über die Kette zu sich selbst zurückführt, bricht den Bau ab.
- `!file` wirkt überall in der Konfiguration, nicht nur unter `custom`. Ein Abruf per URL wartet höchstens 10 Sekunden.

4.5 Beispiel einer vollständigen Konfiguration

Das folgende Beispiel zeigt eine praxisnahe, vollständige `markpublish.yaml`:

```yaml
# markpublish.yaml

document:
  title: "Projektbericht Jahresabschluss"
  subtitle: "Analyse, Ergebnisse und Ausblick"
  summary: "Umfassender Gesamtbericht über die Projektphasen des vergangenen
Geschäftsjahres."
  author: "Projektgruppe Controlling"
  date: "auto"
  version: "1.2.0"
  language: "de"
  copyright: "© 2026 Musterunternehmen GmbH"

  # Layout-Elemente
  cover: true
  header: true
  footer: true

  # Verzeichnistiefen
  document_toc: 2
  part_toc: 2
```

```yaml
  chapter_toc: "none"

  # Nummerierung: Slot 1 ist der Part, Slot 2 das Kapitel, Slot 3 die H2
  autonum_pattern: "_|1|.1|+"

theme: "default"

parts:
  - part: "Hauptteil"
    break_before: "none"
    chapters:
      - file: "chapters/01_einleitung.md"
        break_before: "page"

      - file: "chapters/02_analyse.md"
        # Kurzform für das Inhaltsverzeichnis bei langer Datei-H1:
        toc_title: "Marktanalyse"
        break_before: "page"

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

  - part: "Anhänge"
    break_before: "divider"
    # Am Part beginnt Slot 1 beim Kapitel: '_' lässt es unnummeriert,
    # die H2 darunter zählen ab 1 - je Kapitel neu.
    autonum_pattern: "_|1|.1|+"
    autonum_reset: true
    chapters:
      - file: "chapters/anhang_tabellen.md"

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

Eine vollständige Übersicht aller verfügbaren Schlüssel, Datentypen, Standardwerte und Kombinationsmöglichkeiten für jede der drei Ebenen finden Sie in **Anhang A: Schema-Referenz markpublish.yaml** am Ende dieses Handbuchs.

© 2026 Frank Winter

KAPITEL 5

Templates und Mehrsprachigkeit

> Theme-Struktur, dreistufige Auflösungshierarchie und Textkaskade (i18n).

AUF EINEN BLICK

5.1 Die Auflösungshierarchie für Themes 28

5.2 Aufbau eines Themes 28

5.3 Eigene Themes erstellen und anpassen 30

5.4 Mehrsprachigkeit und Textkaskade (i18n) 31

markpublish trennt Inhalt und Gestaltung strikt voneinander: Die Textinhalte entstehen in Markdown, während das visuelle Erscheinungsbild, die Typografie und das Seitenlayout über Typst-Templates gesteuert werden. Ergänzt wird dieses System durch eine flexible Kaskade zur Mehrsprachigkeit (i18n).

5.1 Die Auflösungshierarchie für Themes

Wenn markpublish nach einem Theme sucht (angegeben über das Kommandozeilen-Flag `--theme`, die Einstellung `theme:` in der Konfigurationsdatei oder den Standard `default`), durchsucht der Resolver folgende Ebenen der Reihe nach (der erste Treffer gewinnt):

1. **Benutzer-Vorlagen (`user`):**
   Vorlagen im Home-Verzeichnis des aktuellen Benutzers (`~/.markpublish/templates/<theme>/`) bzw. im benutzerspezifischen AppData-Verzeichnis. Dies ermöglicht autorenweite Standardvorlagen über alle Projekte hinweg.
2. **Projekt- / Vorlagen-Verzeichnis (`common`):**
   Vorlagen im Projektordner (`./templates/<theme>/`) oder in einem explizit über `templates_dir:` in der `markpublish.yaml`, über das Kommandozeilen-Flag `--templates-dir` oder die Umgebungsvariable `MARKPUBLISH_TEMPLATES_DIR` festgelegten Verzeichnis.
3. **Paket-Vorlagen (`package`):**
   Das mitgelieferte Standard-Theme `default`. Es dient als verlässlicher Fallback, wenn auf Benutzer- oder Projektebene keine Vorlage gefunden wird.

5.2 Aufbau eines Themes

Ein vollständiges markpublish-Theme besitzt folgende Verzeichnisstruktur:

```
templates/
└── mein-theme/
    ├── i18n.yaml            # Übergreifende Beschriftungen des Themes
    └── pdf/
        ├── template.typ     # Typst-Layoutvorlage für das PDF
        ├── i18n.yaml        # Formatspezifische Beschriftungen des Themes
        ├── fonts/           # Optionale Schriftdateien (z. B. .ttf, .otf)
        └── assets/          # Statische Grafiken, Logos und Icons
            └── icons/
```

5.2.1 Der Theme-Contract (`setup-document` und `meta`)

Das Herzstück der Layoutdatei `template.typ` ist die Typst-Funktion `setup-document`. Sie empfängt Layoutschalter, Textbeschriftungen und sämtliche Metadaten aus markpublish:

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

Metadatenübergabe über `meta`
markpublish übergibt alle Dokumentmetadaten geschlossen als strukturiertes Typst-Wörterbuch `meta: (:)`. Jedes Theme muss diesen Parameter in `setup-document` deklarieren (oder `..rest` akzeptieren). Fehlt der Parameter, meldet `markpublish labels` dies noch vor dem Build als Signaturfehler.

Innerhalb des Typst-Templates greifen Sie auf die einzelnen Metadatenfelder über ihren Namen zu:

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

Freie oder optionale Metadatenfelder sollten stets mit einem sicheren Fallback abgefragt werden:

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

Wird im Theme ein Feld ohne `default:` abgefragt, das in der Konfiguration nicht definiert ist, bricht der Build mit einem klaren `UndefinedMetadataError` ab. Dieser nennt die genaue Zeile im Theme und verweist auf die `markpublish.yaml`.

5.2.2 Was auf dem Titelblatt erscheint (`cover-fields`)

Das Metadatenraster des Deckblatts zeigt nicht alles, was unter `document:` steht, sondern genau die Angaben, die das Theme dafür vorsieht – im mitgelieferten Standard-Theme sind das Version, Datum, Autor, Copyright und Status. Titel, Untertitel und Zusammenfassung stehen als Hauptblöcke darüber.

© 2026 Frank Winter

Welche Felder auf dem Deckblatt erscheinen, bestimmt im Theme die Funktion `cover-fields`:

```
#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),
)
```

Wer `abteilung:` oder `kunde:` auf dem Deckblatt haben möchte, exportiert das Theme (siehe unten) und ergänzt dort eine Zeile. Die zugehörige Beschriftung stammt aus der `i18n.yaml` des Projekts.

5.2.3 Typisierte Metadatenwerte

Metadaten behalten ihren YAML-Typ auf dem Weg zu Typst bei: * Zahlen und Zeichenketten bleiben Zahlen und Strings. * Ein `freigegeben: true` kommt als echter Typst-Wahrheitswert an. Das Standard-Theme formatiert diesen automatisch über die Beschriftungsschlüssel `bool_true` („Ja“) bzw. `bool_false` („Nein“) aus der i18n-Kaskade. * Listen werden sauber komma-separiert formatiert.

Welche Felder ein Theme tatsächlich abholt, zeigt `markpublish labels`: Ein Feld, das kein Theme verwendet, erscheint dort mit dem Befund *ungenutzt*.

5.2.4 Abbildungen, Bilder und Verzeichnisse

- Beschriftete Abbildungen und Tabellen kommen als Typst-`figure` an; gestaltet werden sie mit `show figure...`-Regeln.
- Jedes Bild außerhalb des Fließtexts steckt in einer `box` mit der Marke `<mp-image>`, `<mp-image-frame>` (`frame`) oder `<mp-image-noframe>` (`noframe`). Ob `<mp-image>` gerahmt wird, ist der Standard des Themes; im Standard-Theme steuert ihn `image-frame-default` (`true`).
- Ein Eintrag `list_of` ruft `render-list-of(kind:, title:, slug:, in-toc:)` auf. Das Theme muss die Funktion nur definieren, wenn das Dokument `list_of` verwendet.

5.3 Eigene Themes erstellen und anpassen

Der einfachste und sicherste Weg zur Erstellung eines eigenen Corporate Designs ist der Export des integrierten Standard-Themes:

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

Dieser Befehl kopiert das Standard-Theme vollständig in das lokale Verzeichnis `templates/default/`. Sie können die Dateien anschließend direkt bearbeiten, Schriften anpassen, Farben ändern oder Ihr Firmenlogo einbinden. markpublish greift beim nächsten `build` automatisch auf Ihre angepasste Projektvorlage zu.

Die Layoutdatei `template.typ` ist in der Satzsprache Typst geschrieben und im Standard-Theme durchgehend kommentiert. Wer sie über Farben und Schriften hinaus umbauen möchte, findet die vollständige Sprachreferenz unter https://typst.app/docs.

5.4 Mehrsprachigkeit und Textkaskade (i18n)

Professionelle Dokumente enthalten eine Vielzahl statischer Texte, die nicht aus den Markdown-Kapiteln stammen, sondern vom Template erzeugt werden – beispielsweise „Inhaltsverzeichnis“, „Kapitel“, „Seite X von Y“ oder Hinweise im Deckblatt.

markpublish verwaltet diese Texte über ein Kaskadensystem in `i18n.yaml`-Dateien.

5.4.1 Die vierstufige Beschriftungskaskade

Bei der Auflösung eines Textschlüssels (z. B. `toc_title`) sucht markpublish in folgender Reihenfolge – spätere Fundstellen überschreiben frühere:

1. **Paket-Basis (`markpublish/i18n.yaml`):** Vollständige Standardbeschriftungen für alle unterstützten Sprachen.
2. **Theme-Ebene (`<theme>/i18n.yaml`):** Themes können eigene Formulierungen oder Bezeichnungen definieren.
3. **Format-Ebene (`<theme>/pdf/i18n.yaml`):** Formatspezifische Anpassungen des Themes.
4. **Projekt-Ebene (`i18n.yaml` neben der `markpublish.yaml`):** Texte dieses einen Dokuments – mit höchster Priorität.

Die Projekt-Ebene ist der Ort für die Beschriftung eigener Metadatenfelder: Wer unter `document:` ein `abteilung: "F&E"` notiert, schreibt hier `abteilung: "Abteilung"` dazu. Ohne diesen Eintrag druckt das Deckblatt den Schlüssel selbst. Sie können damit auch einen Text des Themes für ein einzelnes Dokument ersetzen, ohne das Theme zu kopieren.

5.4.2 Aufbau der i18n.yaml

Eine `i18n.yaml` enthält strukturierte Übersetzungen je Sprachkürzel:

```yaml
de:
  toc_title: "Inhaltsverzeichnis"
  chapter: "Kapitel"
```

© 2026 Frank Winter

```
  part: "Teil"
  page: "Seite"
  page_of: "von"
  version: "Version"
  author: "Autor"

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

5.4.3 Spracherkennung und Validierung

Die Sprache eines Dokuments wird in der `markpublish.yaml` über den Schlüssel `language:` (z. B. `language: "de"`) festgelegt. Fehlt dieser Eintrag, ermittelt markpublish automatisch die Systemsprache des Autors.

Mit dem CLI-Befehl `markpublish labels` können Sie jederzeit prüfen, welche Beschriftungen für Ihr Projekt aktiv sind und aus welcher Datei sie stammen.

© 2026 Frank Winter

KAPITEL 6

Besondere Markdown-Features

> Auszeichnung und typografische Darstellung von Hinweisboxen, Definitionslisten, Aufgaben und mathematischen Formeln.

AUF EINEN BLICK

6.1 Hinweisboxen (Admonitions und Callouts) 34

6.2 Definitionslisten 36

6.3 Aufgabenlisten (Tasklists) 37

6.4 Fußnoten 37

6.5 Mathematische Formeln 38

6.6 Typografische Textauszeichnungen und Satzzeichen 39

6.7 Automatische Symbole und Pfeile 39

6.8 Bilder, Abbildungen und Tabellen 40

6.9 Automatische Verlinkung und Querverweise 42

6.10 Maskierungsfreie Sonderzeichen 42

markpublish unterstützt den vollen Funktionsumfang des modernen CommonMark- und GitHub-Flavored-Markdown-Standards. Grundlegende Elemente wie Überschriften, Absätze, Textauszeichnungen (*kursiv*, **fett**), Tabellen, Aufzählungen, Quellcode-Blöcke und Weblinks funktionieren exakt wie gewohnt.

Dieses Kapitel konzentriert sich ausschließlich auf die darüber hinausgehenden Spezialfeatures und typografischen Erweiterungen von markpublish. Jeder Abschnitt zeigt zuerst die Auszeichnung im Markdown und unmittelbar darunter das Ergebnis, wie es in diesem Handbuch gesetzt ist.

6.1 Hinweisboxen (Admonitions und Callouts)

Zur optischen Hervorhebung wichtiger Passagen unterstützt markpublish GitHub-konforme Callout-Blöcke. Sie werden als Zitatblock notiert, dessen erste Zeile den Typ der Box in eckigen Klammern nennt:

```
> [!NOTE]
> Dies ist ein allgemeiner Informationstext mit nützlichen Details.

> [!TIP]
> Ein praktischer Tipp für schnellere Arbeitsabläufe.

> [!IMPORTANT]
> Wichtige Information, die für das Verständnis unerlässlich ist.

> [!WARNING]
> Warnung vor möglichen Fehlbedienungen oder unerwartetem Verhalten.

> [!CAUTION]
> Kritischer Sicherheitshinweis vor potenziellen Datenverlusten.
```

Im PDF entstehen daraus gestaltete Boxen mit farbigem Rahmen und Icon:

> **Hinweis**
> Dies ist ein allgemeiner Informationstext mit nützlichen Details.

> **Tipp**
> Ein praktischer Tipp für schnellere Arbeitsabläufe.

© 2026 Frank Winter

> **Wichtig**
> Wichtige Information, die für das Verständnis unerlässlich ist.

> **Warnung**
> Warnung vor möglichen Fehlbedienungen oder unerwartetem Verhalten.

> **Achtung**
> Kritischer Sicherheitshinweis vor potenziellen Datenverlusten.

Wird ein eigener Titel gewünscht, kann dieser direkt hinter dem Typ in der ersten Zeile angegeben werden:

```
> [!NOTE] Individuelle Überschrift
> Eigener Inhalt mit spezifischem Titel.
```

> **Individuelle Überschrift**
> Eigener Inhalt mit spezifischem Titel.

6.1.1 Automatische Lokalisierung über i18n-Labels

Ohne eigenen Titel wird die Titelleiste der Hinweisboxen („Hinweis“, „Tipp“, „Wichtig“, „Warnung“, „Achtung“) automatisch über die Beschriftungsdatei `i18n.yaml` der gewählten Dokumentsprache bezogen:

Box-Typ	Verwendeter i18n-Schlüssel	Deutsche Beschriftung
`[!NOTE]`	`alert_note`	Hinweis
`[!TIP]`	`alert_tip`	Tipp
`[!IMPORTANT]`	`alert_important`	Wichtig
`[!WARNING]`	`alert_warning`	Warnung
`[!CAUTION]`	`alert_caution`	Achtung

© 2026 Frank Winter

6.1.2 Alternative Schreibweise mit `!!!`

Neben der GitHub-Notation versteht markpublish auch die aus MkDocs bekannte Admonition-Schreibweise. Der Typ folgt auf drei Ausrufezeichen, ein optionaler Titel steht in Anführungszeichen dahinter, der Inhalt wird um vier Leerzeichen eingerückt:

```
!!! warning "Eigener Titel"
    Der eingerückte Inhalt darf mehrere Absätze umfassen.
```

> ⚠ **Eigener Titel**
> Der eingerückte Inhalt darf mehrere Absätze umfassen.

Beide Schreibweisen erzeugen dieselbe gestaltete Box. Ein Unterschied bleibt allerdings: Ohne eigenen Titel setzt `!!!` die englische Vorgabe der Erweiterung („Note“, „Tip“, „Warning“) statt der Beschriftung aus der i18n-Kaskade. Wer die lokalisierte Titelzeile möchte, notiert entweder `> [!NOTE]` oder gibt bei `!!!` einen ausdrücklich leeren Titel an:

```
!!! note ""
    Die Titelzeile stammt dann aus der i18n-Kaskade.
```

> ⓘ **Hinweis**
> Die Titelzeile stammt dann aus der i18n-Kaskade.

Ein unbekannter Typ (etwa `!!! info`) wird als Hinweis-Box gesetzt und behält seinen Namen als Titelzeile.

6.2 Definitionslisten

Für Glossare, Begriffserklärungen oder Parameterübersichten bieten Definitionslisten eine besonders saubere typografische Struktur. Der Begriff steht in einer eigenen Zeile, die zugehörige Erläuterung folgt darunter, eingeleitet durch einen Doppelpunkt:

```
Markdown
: Einfache, lesbare Auszeichnungssprache für formatierte Texte im
Klartextformat.

markpublish.yaml
: Zentrale Projekt-Konfigurationsdatei zur Steuerung von Dokumentstruktur,
Metadaten und Nummerierung.
```

© 2026 Frank Winter

Gesetzt wird daraus ein Block aus hervorgehobenem Begriff und eingerückter Erläuterung:

Markdown Einfache, lesbare Auszeichnungssprache für formatierte Texte im Klartextformat.

markpublish.yaml Zentrale Projekt-Konfigurationsdatei zur Steuerung von Dokumentstruktur, Metadaten und Nummerierung.

6.3 Aufgabenlisten (Tasklists)

Zur Darstellung von Checklisten, To-do-Listen oder Meilensteinen stehen Aufgabenlisten zur Verfügung. Notiert werden sie als Aufzählung, deren Einträge mit einer leeren oder angekreuzten Klammer beginnen:

```
- [x] Python 3.10 oder neuer bereitstellen
- [x] markpublish installieren
- [ ] Erstes eigenes Dokument veröffentlichen
```

markpublish setzt Aufgabenlisten typografisch sauber ohne vorangestellte Aufzählungspunkte (Bullets) direkt mit der Checkbox und dem Text; mehrzeilige Beschreibungen brechen bündig unter der ersten Zeile um:

- [x] Python 3.10 oder neuer bereitstellen
- [x] markpublish installieren
- [ ] Erstes eigenes Dokument veröffentlichen

6.4 Fußnoten

markpublish unterstützt die klassische Markdown-Notation für Fußnoten. Damit lassen sich weiterführende Hinweise, Quellenangaben oder detaillierte Erläuterungen auslagern, ohne den Lesefluss des Haupttextes zu unterbrechen.

Die Notation erfolgt zweistufig: An der gewünschten Textstelle steht ein Verweis wie `[^1]` oder ein sprechender Bezeichner wie `[^hinweis]`. Die zugehörige Definition kann an beliebiger Stelle im Markdown-Dokument notiert werden – üblicherweise am Ende des jeweiligen Abschnitts oder Kapitels:

```
Dieser Satz enthält eine nummerierte Fußnote[^1] sowie eine Notiz mit Text-
Schlüssel[^hinweis].
```

© 2026 Frank Winter

```
[^1]: Dies ist der Inhalt der ersten Fußnote.
[^hinweis]: Text-Schlüssel werden bei der Ausgabe automatisch in die richtige
Ziffer umgewandelt.
```

Im Satz ergibt das die folgenden beiden Verweise; ihre Texte stehen am unteren Rand dieser Seite, und Verweisziffer und Fußnotentext sind im PDF gegenseitig verlinkt:

Dieser Satz enthält eine nummerierte Fußnote[1] sowie eine Notiz mit Text-Schlüssel[2].

6.5 Mathematische Formeln

markpublish ermöglicht das direkte Setzen mathematischer Formeln im Fließtext oder als abgesetzte Formelblöcke. Die Formeln werden über die Satz-Engine Typst typografisch präzise gerendert.

Unterstützt werden gängige mathematische Ausdrücke wie Brüche (`\frac{a}{b}`), Wurzeln (`\sqrt{x}`), Summen (`\sum`), Integrale (`\int`), griechische Symbole (`\alpha`, `\pi`, `\sigma` etc.) sowie die native Typst-Math-Syntax. Formeln im Fließtext stehen zwischen einfachen Dollarzeichen, abgesetzte Formelblöcke zwischen doppelten:

```
Die berühmte Energie-Masse-Äquivalenz lautet $E = m c^2$.

Abgesetzte Kreisflächenberechnung und quadratische Lösungsformel:

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

Im Satz ergibt das:

Die berühmte Energie-Masse-Äquivalenz lautet $E = mc^2$.

Abgesetzte Kreisflächenberechnung und quadratische Lösungsformel:

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

Typografischer Hinweis zu Variablen:
In Typst Math stehen mehrbuchstabige Wörter für Funktionen oder Bezeichner (wie `sin`, `cos`, `sqrt`). Die Multiplikation einzelner Variablen wird deshalb durch ein Leerzeichen getrennt notiert (z. B. `m c^2` oder `4 a c`).

---

[1] Dies ist der Inhalt der ersten Fußnote.
[2] Text-Schlüssel werden bei der Ausgabe automatisch in die richtige Ziffer umgewandelt.

© 2026 Frank Winter

6.6 Typografische Textauszeichnungen und Satzzeichen

markpublish unterstützt über Standard-Markdown hinaus feine typografische Textauszeichnungen und echte typografische Satzzeichen:

Auszeichnung	Eingabe	Ausgabe	Typischer Einsatzzweck
Hochstellung	`10^3^` oder `m^2^`	10^3 bzw. m^2	Potenzen, Quadrat- und Kubikmeter
Tiefstellung	`H~2~O` oder `CO~2~`	H_2O bzw. CO_2	Chemische Formeln, Indizes
Durchgestrichen	`~~veraltet~~`	~~veraltet~~	Korrekturen, überholte Angaben
Halbgeviertstrich (En-Dash)	`10--20 Uhr`	10–20 Uhr	Bis-Striche, Zeit- und Seitenspannen
Geviertstrich (Em-Dash)	`Einwurf --- oder nicht`	Einwurf — oder nicht	Gedanklicher Einschub, Sprechpause
Auslassungspunkte (Ellipse)	`Warten...`	Warten…	Satzabbrüche, Auslassungen

> **Fließtext und mathematischer Formelsatz**
> Die Auszeichnungen `^hochgestellt^` und `~tiefgestellt~` eignen sich ideal für gebräuchliche Einheiten (m^2) und chemische Formeln (H_2O) im Fließtext. Für mathematische Gleichungen und Variablenformeln empfiehlt sich die native Formelumgebung mit `$...$` bzw. `$$...$$` (siehe Abschnitt *Mathematische Formeln*).

6.7 Automatische Symbole und Pfeile

Häufige Symbole und Operatoren werden bei der Eingabe im Fließtext automatisch in typografische Glyphen umgewandelt:

© 2026 Frank Winter

Symbol	Eingabe	Ausgabe	Typischer Einsatzzweck
Pfeile	`-->`, `<--`, `<-->`	→, ←, ↔	Ablaufschritte, Rückverweise, Äquivalenz
Brüche	`1/2`, `1/4`, `3/4`	½, ¼, ¾	Mengenangaben, Brüche
Rechenzeichen	`+/-`, `=/=`	±, ≠	Toleranzen (±), mathematische Ungleichheit (≠)
Urheberrecht & Marken	`(c)`, `(tm)`, `(r)`	©, ™, ®	Copyright, Trademark, Registered

6.8 Bilder, Abbildungen und Tabellen

Bildattribute stehen in geschweiften Klammern direkt hinter dem Bild:

```
![Umsatz](img/umsatz.png){width=60%}
![Logo](img/logo.png){width=4cm noframe}
```

Attribut	Werte	Wirkung
`width`, `height`	`60%`, `4cm`, `12mm`, `100pt`, `300` (Pixel)	Größe des Bildes
`frame`, `noframe`	–	Rahmen ein bzw. aus, abweichend vom Theme; auch mit Punkt (`.frame`)

Das Seitenverhältnis bleibt immer erhalten: Mit nur einer Angabe ergibt sich die andere von selbst; mit beiden wird das Bild in diese Fläche eingepasst, ohne verzerrt oder beschnitten zu werden.

Ob Bilder gerahmt werden, gibt das verwendete Theme vor; `frame` und `noframe` weichen im Einzelfall davon ab. Das Standard-Theme rahmt jedes Bild, das allein im Absatz steht – `noframe` passt dort etwa für Logos oder freigestellte Grafiken.

Eine Beschriftung folgt als Block unter dem Bild bzw. der Tabelle. Nummer und Wort setzt markpublish selbst:

```
![markpublish-Logo](img/markpublish_logo.png){width=3cm}

/// figure-caption
    attrs: {id: fig-logo}
Das Logo von markpublish
///
```

© 2026 Frank Winter

```
Text zwischen Abbildung und Tabelle: Abbildungen und Tabellen setzt das
Standard-Theme mit etwas mehr Abstand vom Fließtext ab als zwei Absätze
voneinander, ihre Beschriftung steht dagegen eng am Bild bzw. an der Tabelle.

| Format | Beschriftung | Verweis |
| :--- | :--- | :--- |
| Abbildung | `/// figure-caption` | `[](#fig-logo)` |
| Tabelle | `/// table-caption` | `[](#tbl-beschriftung)` |

/// table-caption
    attrs: {id: tbl-beschriftung}
Beschriftungen und Verweise
///
```

Ergebnis:

Abbildung 1: Das Logo von markpublish

Text zwischen Abbildung und Tabelle: Abbildungen und Tabellen setzt das Standard-Theme mit etwas mehr Abstand vom Fließtext ab als zwei Absätze voneinander, ihre Beschriftung steht dagegen eng am Bild bzw. an der Tabelle.

Tabelle 1: Beschriftungen und Verweise

Format	Beschriftung	Verweis
Abbildung	`/// figure-caption`	`[](#fig-logo)`
Tabelle	`/// table-caption`	`[](#tbl-beschriftung)`

Ein Link ohne Text auf die `id` wird zum Verweis mit Nummer: `[](#fig-logo)` ergibt Abbildung 1, `[](#tbl-beschriftung)` ergibt Tabelle 1. Mit Text (`[siehe oben](#fig-logo)`) bleibt es ein gewöhnlicher Link.

Die Wörter kommen aus der i18n-Kaskade (`figure`, `table`). Abbildungs- und Tabellenverzeichnis werden in der `markpublish.yaml` platziert (`list_of`, siehe Schema-Referenz).

© 2026 Frank Winter

6.9 Automatische Verlinkung und Querverweise

Web- und Mailadressen werden bei direkter Eingabe automatisch erkannt und formatiert. Über Anker-Links lässt sich zudem auf jede Überschrift im Dokument verweisen:

Funktion	Eingabe	Ausgabe	Erläuterung
Web-Adresse (Magic Link)	`https://example.com`	https://example.com	URLs werden ohne Klammern verlinkt
E-Mail-Adresse (Magic Link)	`kontakt@example.com`	kontakt@example.com	Mailadressen werden zu klickbaren `mailto:`-Links
Dokument-Querverweis	`[Kapitelanfang](#besondere-markdown-features)`	Kapitelanfang	Klickbare Sprungmarke auf Überschriften
Verweis in einen Anhang	`[](#schema-chapters)`	Teil „Anhänge“, Anhang A, Abschnitt A.4 „Kapitel-Ebene (chapters)“	Anderer Teil: alle Ebenen ab dem Teil
Verweis im selben Kapitel	`[](#abbildungen)`	Abschnitt 6.8 „Bilder, Abbildungen und Tabellen“	Nur der Abschnitt

Ein Link ohne Text auf eine Überschrift wird zum nummerierten Verweis, auch über Dateien hinweg; Wort, Nummer und Titel setzt markpublish. Genannt werden die Ebenen ab der ersten, in der sich Verweisstelle und Ziel unterscheiden. Eine Ebene ohne Nummer erscheint mit ihrem Titel – hier der Teil „Anhänge“.

Für Verweise lohnt eine feste `id` an der Überschrift (`## Restrisiken {#restrisiken}`): die automatische folgt dem Text und ändert sich mit ihm. Ein Verweis auf eine unbekannte `id` bricht den Build ab. Die Wörter kommen aus der i18n-Kaskade (`part`, `chapter`, `section`, `quote_open`, `quote_close`) bzw. aus `label` und `chapter_label`.

6.10 Maskierungsfreie Sonderzeichen

Viele Dokumentationswerkzeuge erfordern das umständliche Maskieren bestimmter Symbole. markpublish verarbeitet gebräuchliche Zeichen im Fließtext vollautomatisch und kollisionsfrei:

© 2026 Frank Winter

Kategorie	Beispiel	Verhalten im Dokument
Währungsbeträge	`$5.00` oder `10.50 $`	Werden nicht versehentlich als mathematische Formeln interpretiert.
Programmiersprachen	`C#`	Löst keine internen Typst-Sonderbefehle aus.
Benutzernamen & Erwähnungen	`@author` oder `user@domain.com`	Werden sicher als Text gedruckt bzw. verlinkt.
Technische Bezeichner	`<zielverzeichnis>`	Spitze Klammern bleiben als Text erhalten und gehen nicht verloren.

© 2026 Frank Winter

KAPITEL 7

Daten im Text

> Platzhalter, Schleifen, Kurznamen, Nachschlagen und Bedingungen für Werte aus der markpublish.yaml.

AUF EINEN BLICK

7.1 Platzhalter 45

7.2 Schleifen 45

7.3 Kurznamen 46

7.4 Nachschlagen 46

7.5 Bedingungen 47

7.6 Regeln für Anweisungen 47

7.7 Beispiel: Datenflüsse und Systeme 47

Werte aus der `markpublish.yaml` lassen sich in den Text einsetzen, wiederholen, nachschlagen und an Bedingungen knüpfen. Anweisungen wertet markpublish vor dem Markdown aus, eingesetzt werden die Werte erst danach: Ein `|` oder `*` in einem Wert kann deshalb keine Tabelle und keine Auszeichnung zerstören.

Grundsatz: Kein Wert darf fehlen. Ein Platzhalter oder Pfad ohne Wert bricht den Bau mit Datei und Zeile ab, statt eine Lücke zu lassen.

7.1 Platzhalter

`{{name}}` setzt eine Angabe aus `document:` ein, etwa `{{author}}`, `{{version}}` oder `{{date}}` (bei `"auto"` das errechnete Datum); das gilt auch für eigene Metadatenfelder. Werte, die nur im Text gebraucht werden, gehören unter `custom:`. Dort dürfen sie verschachtelt sein und werden mit Punkten angesprochen:

```yaml
document:
  author: "Frank Winter"
  custom:
    support_mail: "support@example.com"
    verfahren:
      name: "Verfahren X"
```

```
Ansprechpartner: {{author}}, {{custom.support_mail}}.
Dieses Dokument beschreibt {{custom.verfahren.name}}.
```

Ist ein eingesetzter Wert eine E-Mail-Adresse oder eine URL, wird er ein Link; steht der Platzhalter schon in einem Link, bleibt es bei diesem. Werte unter `custom` erreichen das Theme nicht. Größere Datenbestände lassen sich mit `!file` aus eigenen Dateien laden (siehe Kapitel 4, Abschnitt 4.4 „Werte aus anderen Dateien (!file)“).

7.2 Schleifen

`{% for %}` wiederholt die Zeilen bis `{% endfor %}` für jeden Eintrag einer Gruppe, in der Reihenfolge der YAML-Datei – etwa eine Tabellenzeile je Rolle:

```yaml
document:
  custom:
    rollen:
      owner: {bezeichnung: "Verfahrensverantwortliche/r", name: "A. Muster"}
      admin: {bezeichnung: "Technische Administration", name: "B. Beispiel"}
```

© 2026 Frank Winter

```
| Rolle | Name | Schlüssel |
| :--- | :--- | :--- |
{% for (rolle, key) in custom.rollen %}
| {{rolle.bezeichnung}} | {{rolle.name}} | {{key}} |
{% endfor %}
```

`(rolle, key)` liefert zusätzlich den Namen des Eintrags (`owner`); `for rolle in …` genügt, wenn er nicht gebraucht wird. Schleifen lassen sich schachteln, ihr Name gilt nur in ihnen.

7.3 Kurznamen

`{% set %}` gibt einem langen Pfad einen Kurznamen. Er gilt bis zum Ende der Datei, innerhalb einer Schleife oder Bedingung bis zu deren Ende:

```
{% set owner = custom.rollen.owner %}
Verantwortlich ist {{owner.name}}.
```

7.4 Nachschlagen

Verweist ein Wert per Schlüssel auf einen Eintrag einer anderen Gruppe, schlägt `gruppe[pfad]` ihn nach: markpublish setzt den Wert von `pfad` als Schlüssel ein. So stehen Daten nur an einer Stelle, auch wenn mehrere Einträge sie brauchen.

```yaml
document:
  custom:
    systeme:
      crm: {name: "Salesforce CRM", standort: "EU (Frankfurt)", extern: true}
    datenfluesse:
      kunden: {bezeichnung: "Kundenstammdaten", system: crm}
```

`{{custom.systeme[custom.datenfluesse.kunden.system].name}}` ergibt `Salesforce CRM`. In Schleifen und mit Kurznamen wird es handlich:

```
{% for d in custom.datenfluesse %}
{% set sys = custom.systeme[d.system] %}
- {{d.bezeichnung}}: {{sys.name}}, {{sys.standort}}
{% endfor %}
```

- Der Wert in den Klammern muss ein einzelner Wert sein; ein Schleifenschlüssel (`key`) geht ebenso.

© 2026 Frank Winter

- Gibt es keinen Eintrag mit diesem Schlüssel, bricht der Bau ab – ein Tippfehler in den Daten fällt so sofort auf.
- Nachschlagen funktioniert im Markdown, nicht in den Angaben unter `document:` selbst (Titel, Deckblatt).

7.5 Bedingungen

`{% if %}` nimmt Zeilen nur auf, wenn die Bedingung zutrifft; `{% else %}` ist optional:

```
{% if custom.verfahren.dsfa_erforderlich %}
Eine Datenschutz-Folgenabschätzung liegt bei (Anlage 3).
{% else %}
Eine Datenschutz-Folgenabschätzung ist nicht erforderlich.
{% endif %}

{% if custom.verfahren.betrieb == "cloud" %}
Der Betrieb erfolgt beim Dienstleister.
{% endif %}
```

- `{% if pfad %}` verlangt einen Wahrheitswert (`true` oder `false`).
- `{% if pfad == "wert" %}` vergleicht einen einzelnen Wert mit einem Text; Zahlen werden als Text verglichen (`== "2024"`).
- Ein fehlender Pfad ist ein Fehler, keine falsche Bedingung.

7.6 Regeln für Anweisungen

- **Anweisungen stehen allein in ihrer Zeile.** Die Zeile verschwindet beim Bau vollständig, eine Tabelle bleibt also zusammenhängend. In einem Zitat oder einer Hinweisbox darf `>` davorstehen.
- **Wörtlich zeigen:** In Code und Codeblöcken bleiben Platzhalter und Anweisungen stehen, im Fließtext mit vorangestelltem Backslash: `\{{author}}`, `\{% … %}`.
- **Namen:** Ein Kurz- oder Schleifenname darf keine Angabe unter `document` verdecken (`author`, `custom` …).
- **Fehler** wie ein fehlendes `{% endfor %}` oder `{% endif %}`, ein unbekannter Pfad oder ein falsch verschachtelter Block brechen den Bau mit Datei und Zeile ab.

7.7 Beispiel: Datenflüsse und Systeme

Die `markpublish.yaml` dieses Handbuchs enthält eine kleine Datensammlung:

© 2026 Frank Winter

```yaml
document:
  custom:
    systeme:
      crm: {name: "Salesforce CRM", standort: "EU (Frankfurt)", extern: true}
      erp: {name: "SAP S/4HANA", standort: "Deutschland", extern: false}
    datenfluesse:
      kunden: {bezeichnung: "Kundenstammdaten", system: crm}
      rechnungen: {bezeichnung: "Rechnungsdaten", system: erp}
```

Eine Tabelle mit einer Zeile je Datenfluss, das System nachgeschlagen und der Hinweis abhängig vom Betrieb:

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

Ergebnis:

Datenfluss	System	Standort	Betrieb
Kundenstammdaten	Salesforce CRM	EU (Frankfurt)	Dienstleister
Rechnungsdaten	SAP S/4HANA	Deutschland	eigenes Rechenzentrum

Ändert sich der Standort eines Systems, genügt eine Zeile in der YAML-Datei – jede Tabelle, die es nachschlägt, zieht mit.

© 2026 Frank Winter

TEIL

Anhänge

Vollständige Spezifikations-Tabellen und Hilfestellungen zur Fehlerbehebung.

INHALT DIESES TEILS

Anhang A: Schema-Referenz markpublish.yaml **50**

A.1 Dokument-Ebene (document) 50

A.2 Globale Projekt-Einstellungen 52

A.3 Abschnitts-Ebene (parts) 53

A.4 Kapitel-Ebene (chapters) 54

A.5 Verzeichnis-Steuerung (Scope-Werte) 56

Anhang B: Fehlerbehebung und FAQ **57**

B.1 Diagnose von Kompilierungsfehlern 57

B.2 Häufige Ursachen und Lösungen 57

B.3 Häufig gestellte Fragen (FAQ) 60

Anhang A: Schema-Referenz markpublish.yaml

Dieser Anhang enthält die vollständige Spezifikation aller Konfigurationsoptionen der `markpublish.yaml`.

A.1 Dokument-Ebene (document)

Die Sektion `document:` legt globale Metadaten, Layoutschalter, Verzeichnisvorgaben und Nummerierungsstile für das Gesamtdokument fest.

Schlüssel	Typ	Default	Beschreibung
`title`	String	*(Pflichtfeld)*	Titel des Dokuments (erscheint auf Deckblatt, Kopfzeilen und Metadaten).
`subtitle`	String	`None`	Untertitel des Dokuments.
`summary`	String	`None`	Zusammenfassung oder Abstract (erscheint auf dem Deckblatt).
`author`	String	`None`	Name des Autors oder der Organisation.
`status`	String	`None`	Status des Dokuments (z. B. „Entwurf“, „Freigegeben“, „Vertraulich“).
`copyright`	String	`None`	Copyright-Hinweis (z. B. „© 2026 Frank Winter“).
`date`	String	`"auto"`	Datum des Dokuments. Bei `"auto"` oder `"today"` wird das aktuelle Tagesdatum eingesetzt.
`version`	String	`None`	Versionskennung (z. B. `"2.0.0"`). Wird nur gedruckt, wenn explizit gesetzt.
`language`	String	*(Systemsprache)*	ISO-Sprachcode (z. B. `"de"` oder `"en"`). Steuert Silbentrennung und UI-Texte.
`custom`	Mapping	`{}`	Eigene Werte für Platzhalter im Text (`{{custom.a.b}}`), auch verschachtelt.

Schlüssel	Typ	Default	Beschreibung
			Siehe Kapitel 7 „Daten im Text“. Einzelne Werte oder Gruppen lassen sich mit `!file` nachladen.
`cover`	Boolean	`false`	Steuert, ob eine gestaltete Deckblattseite erzeugt wird. Ohne Angabe beginnt das Dokument mit dem ersten Inhalt.
`header`	Boolean	`true`	Aktiviert oder deaktiviert die laufende Kopfzeile im Dokument.
`footer`	Boolean	`true`	Aktiviert oder deaktiviert die laufende Fußzeile (inkl. Seitennummerierung).
`document_toc`	Scope	`"full"`	Tiefe des Haupt-Inhaltsverzeichnisses (`"none"`, `"full"` oder Zahl ≥ 1).
`part_toc`	Scope	`"full"`	Standardtiefe für lokale Inhaltsverzeichnisse auf Abschnitts-Trennseiten.
`chapter_toc`	Scope	`"full"`	Standardtiefe für lokale Inhaltsverzeichnisse auf Kapitel-Trennseiten.
`autonum_pattern`	String	*s. unten*	Aufbau der Nummern; Slot 1 ist der Part, Slot 2 das Kapitel. `none` schaltet die Nummerierung ab.
`autonum_reset`	Boolean	`false`	Lässt die Ebenen darunter beim Betreten wieder bei 1 anfangen.
`part_label`	String	aus i18n	Wort, das einen Abschnitt benennt (z. B. `"Teil"`). Leer notiert (`""`) entfällt es.
`chapter_label`	String	aus i18n	Wort, das ein Kapitel benennt (z. B. `"Kapitel"`). Leer notiert (`""`) entfällt es.
`pagenum_reset`	Boolean	`false`	Startet die Seitennummerierung bei jedem Abschnitt oder Kapitel neu bei Seite 1.

© 2026 Frank Winter

> **Freie Metadatenfelder unter `document:`**
> Über die aufgeführten Standardfelder hinaus sind unter `document:` beliebige eigene Metadatenfelder erlaubt (z. B. `abteilung: "F&E"`, `freigegeben: true`). Sie werden typisiert an das Theme (`meta: (:)`) übergeben.
>
> Statische Textbeschriftungen (`document.i18n` oder `document.labels`) sind hier jedoch unzulässig und führen zum Abbruch – Textübersetzungen gehören ausschließlich in die `i18n.yaml`-Dateien der Kaskade.

A.2 Globale Projekt-Einstellungen

Direkt auf oberster Ebene der `markpublish.yaml` (neben `document:` und `parts:`) können folgende Schalter gesetzt werden:

Schlüssel	Typ	Default	Beschreibung
`theme`	String	`"default"`	Name des zu verwendenden Themes.
`templates_dir`	String	`None`	Optionaler benutzerdefinierter Pfad zu einem Verzeichnis mit Themes.

> **Strikte Schlüsselprüfung auf oberster Ebene**
> Auf oberster Ebene der `markpublish.yaml` sind ausschließlich die deklarierten Schlüssel (`document`, `theme`, `templates_dir`, `parts`) zulässig. Unbekannte Schlüssel (wie Tippfehler bei `theam:`) werden mit einem Namensvorschlag strikt abgewiesen.

A.2.1 Nummerierungs-Pattern

`autonum_pattern` beschreibt den Aufbau der Nummern als Kette von Slots, getrennt durch `|`. Slot 1 gehört zur ersten Ebene *unter* der Stelle, an der das Pattern steht: unter `document:` ist das der Part, an einem Part das Kapitel, an einem Kapitel die H2. Ein Pattern beschreibt also nie die Ebene, an der es notiert ist.

Symbol	Ergebnis
`1`	1, 2, 3 …
`01`, `001`	01, 02 … bzw. 001, 002 … (feste Stellenzahl)
`a` / `A`	a, b, c … / A, B, C …

© 2026 Frank Winter

Symbol	Ergebnis
`i` / `I`	i, ii, iii … / I, II, III …
`_`	Ebene bleibt unnummeriert
`+`	wiederholt den Slot davor für alle tieferen Ebenen

Alles andere im Slot ist Literal und braucht keine Anführungszeichen; nur wer ein reserviertes Zeichen wörtlich meint, setzt es in Hochkommas (`'Artikel '1`). Ein fehlerhaftes Pattern bricht den Build ab.

```
"_|1|.1|+"      Part ohne Nummer, Kapitel 1, Abschnitt 1.1   (Vorgabe)
"I|1|.1|+"      Abschnitt I, II … Kapitel zählen durch
"'Anhang 'A|.1" Anhang A, Anhang A.1
```

Notiert man ein `label`, tritt das Wort vor die Nummer der Überschrift und ihres Verzeichniseintrags: aus `A` wird `Anhang A:`. Die Unterüberschriften bleiben davon unberührt und zählen weiter `A.1`, `A.2` — deshalb gehört das Wort in diesen Schlüssel und nicht ins Pattern. Das Trennzeichen kommt aus der i18n-Kaskade (`label_separator`).

Die Part-Nummer geht nicht in die Kapitelnummern ein: sie steht auf der Trennseite und im Inhaltsverzeichnis, nicht vor jedem Kapitel. Ein Abschnitt mit `document_toc: "none"` bekommt keine Nummer und verbraucht auch keine.

A.3 Abschnitts-Ebene (parts)

Die Liste `parts:` unterteilt das Dokument in übergeordnete Abschnitte. Ein Abschnitt fasst Kapitel zusammen und vererbt seine Einstellungen nach unten.

Schlüssel	Typ	Default	Beschreibung
`part`	String	*(Pflichtfeld)*	Name des Abschnitts (z. B. „Hauptteil“, „Anhänge“).
`toc_title`	String	`None`	Optionaler abweichender Titel für das Inhaltsverzeichnis und Kopfzeilen (Standard: `part`).
`divider_title`	String	`None`	Optionaler abweichender Titel auf der Abschnitts-Trennseite (Standard: `part`).
`subtitle`	String	`None`	Untertitel des Abschnitts (erscheint auf der Trennseite).

© 2026 Frank Winter

Schlüssel	Typ	Default	Beschreibung
`summary`	String	`None`	Kurzbeschreibung des Abschnitts (erscheint auf der Trennseite).
`break_before`	String	`"none"`	Umbruchverhalten vor dem Abschnitt: `"divider"` (Trennseite), `"page"` (Überschrift auf neuer Seite) oder `"none"` – der Abschnitt gliedert dann nur die Konfiguration und belegt keine eigene Seite.
`document_toc`	Scope	`None`	Überschreibt den Beitrag dieses Abschnitts zum Haupt-Inhaltsverzeichnis.
`part_toc`	Scope	`None`	Steuert das lokale Inhaltsverzeichnis auf der Abschnitts-Trennseite.
`chapter_toc`	Scope	`None`	Vererbt die Vorgabe für Kapitelverzeichnisse an alle Kapitel des Abschnitts.
`autonum_pattern`	String	`None`	Nummern für diesen Abschnitt; Slot 1 ist hier das Kapitel.
`autonum_reset`	Boolean	`None`	Lässt die Kapitel dieses Abschnitts wieder bei 1 anfangen.
`label`	String	aus i18n	Wort, das diesen Abschnitt benennt. Leer notiert (`""`) entfällt es.
`chapter_label`	String	geerbt	Wort für jedes Kapitel dieses Abschnitts (z. B. `"Anhang"`). Leer notiert (`""`) entfällt es.
`pagenum_reset`	Boolean	`None`	Setzt den Seitenzähler zu Beginn des Abschnitts auf 1 zurück.
`chapters`	Liste	*(Pflichtfeld)*	Liste der Inhaltskapitel innerhalb dieses Abschnitts (mindestens 1 Eintrag).

A.4 Kapitel-Ebene (chapters)

Die Liste `chapters:` definiert die Inhaltsdateien. Kapitel stehen immer unmittelbar unter einem Eintrag aus `parts:`.

Die Überschrift auf der Inhaltsseite wird standardmäßig durch die führende `#`-Überschrift der Markdown-Datei bestimmt (kann über `show_title: false` unterdrückt werden). Für Verzeichnisse und Trennseiten stehen zwei explizite, optionale Schlüssel bereit:

© 2026 Frank Winter

Schlüssel	Typ	Default	Beschreibung
`file`	String	`None`	Relativer Pfad zur Markdown-Datei (z. B. `"chapters/01_intro.md"`).
`show_title`	Boolean	`true`	Steuert, ob die `#`-Überschrift der Datei auf der Inhaltsseite gedruckt wird.
`toc_title`	String	`None`	Titel für das Inhaltsverzeichnis (Haupt- und Part-TOC) sowie Kopfzeilen. Standard: Datei-H1.
`divider_title`	String	`None`	Titel auf der Kapitel-Trennseite (`break_before: "divider"`). Standard: Datei-H1.
`subtitle`	String	`None`	Untertitel des Kapitels (erscheint auf Trennseiten).
`summary`	String	`None`	Kurzbeschreibung (wird auf Trennseiten genutzt).
`break_before`	String	`"page"`	Umbruch vor dem Kapitel: `"page"` (neue Seite), `"divider"` (Trennseite) oder `"none"`.
`document_toc`	Scope	`None`	Beitrag dieses Kapitels zum Hauptverzeichnis (`"none"`, `"full"`, Zahl).
`chapter_toc`	Scope	`None`	Lokales Verzeichnis auf der Trennseite dieses Kapitels.
`autonum_pattern`	String	`None`	Nummern innerhalb dieses Kapitels; Slot 1 ist hier die H2. Die Kapitelnummer selbst kommt vom Abschnitt.
`autonum_reset`	Boolean	`None`	Lässt die Überschriften dieses Kapitels wieder bei 1 anfangen.
`label`	String	geerbt	Wort, das dieses Kapitel benennt (z. B. `"Exkurs"`). Leer notiert (`""`) entfällt es.
`pagenum_reset`	Boolean	`None`	Setzt die Seitennummerierung zu Beginn dieses Kapitels auf 1 zurück.
`list_of`	String	`None`	Setzt statt einer Datei ein Verzeichnis: `"figures"` (Abbildungen) oder `"tables"` (Tabellen). Titel aus der i18n-Kaskade oder `toc_title`; ohne Nummer, ohne Trennseite.

```yaml
chapters:
  - list_of: figures
```

© 2026 Frank Winter

```
  - list_of: tables
    toc_title: "Verzeichnis der Tabellen"
```

> **Wichtig**
> Auf `parts:` und `chapters:` sind **nur** die hier aufgeführten Schlüssel erlaubt. Ein unbekannter Schlüssel bricht den Build ab und nennt, falls vorhanden, den ähnlich geschriebenen – `break_befor` also, bevor das Kapitel still mit dem falschen Umbruch gesetzt wird. Freie Felder gibt es ausschließlich unter `document:`; dort erreichen sie das Theme, hier blieben sie wirkungslos.

A.5 Verzeichnis-Steuerung (Scope-Werte)

Die Verzeichnisschalter `document_toc`, `part_toc` und `chapter_toc` akzeptieren folgende Werte:

Wert	Bedeutung
`"none"`	Schaltet das jeweilige Verzeichnis komplett ab bzw. nimmt das Element nicht auf.
`"full"`	Nimmt alle vorhandenen Überschriftenebenen ohne Tiefenbegrenzung auf.
Ganze Zahl (z. B. `1`, `2`, `3`)	Begrenzt die Verzeichnistiefe exakt auf die angegebene Zahl von Ebenen.

> **Keine Booleans zulässig**
> Wahrheitswerte (`true` oder `false`) sind für Verzeichnisschalter unzulässig und führen zu einem `ConfigurationError`. Um ein Verzeichnis zu unterdrücken, notieren Sie stets `"none"`.

© 2026 Frank Winter

Anhang B: Fehlerbehebung und FAQ

Dieser Anhang bietet konkrete Hilfestellungen bei typischen Problemen während des Veröffentlichungsprozesses sowie Antworten auf häufige Fragen.

B.1 Diagnose von Kompilierungsfehlern

Sollte ein Kompiliervorgang mit einer Fehlermeldung der Typst-Engine abbrechen, speichert markpublish den vollständig generierten Typst-Quellcode automatisch im aktuellen Arbeitsverzeichnis:

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

B.1.1 Vorgehen zur Fehleranalyse

1. Öffnen Sie die Datei `.markpublish/last_failed_build.typ` in einem beliebigen Texteditor.
2. Suchen Sie nach dem in der Fehlermeldung genannten Begriff, Variablennamen oder Textfragment.
3. Anhand des umgebenden Kontexts lässt sich unmittelbar erkennen, welches Markdown-Kapitel oder welcher Formatierungsblock die Ursache war.

B.2 Häufige Ursachen und Lösungen

B.2.1 Bilder oder Grafiken werden nicht gefunden

Problem: Typst bricht mit einer Meldung wie `image not found` ab.

Lösung:

- Pfade zu Abbildungen im Markdown (z. B. `![Diagramm](images/architektur.png)`) werden immer **relativ zur jeweiligen Markdown-Datei** aufgelöst.
- Liegt die Datei `01_kapitel.md` im Ordner `chapters/`, muss der Ordner `images/` entweder in `chapters/images/` liegen oder als `../images/architektur.png` referenziert werden.
- markpublish kopiert lokale Bilddateien während des Builds automatisch in den internen Build-Ordner.

© 2026 Frank Winter

B.2.2 Ungültige Einrückungen in der Konfiguration (YAML)

Problem: markpublish meldet beim Start `Configuration error: mapping values are not allowed here` oder `YAML parsing error`.

Lösung:

- YAML reagiert strikt auf falsche Einrückungen. Verwenden Sie für Einrückungen stets **zwei Leerzeichen** und niemals Tabulatoren.
- Achten Sie darauf, dass Listeneinträge (`-`) und verschachtelte Schlüssel bündig zueinander stehen.

B.2.3 Anpassung statischer Textbeschriftungen (i18n)

Problem: Ein generierter Text (wie „Inhaltsverzeichnis“ oder „Kapitel“) soll umformuliert werden oder fehlt in einer neuen Sprache.

Lösung:

- Führen Sie `markpublish labels` aus, um alle aufgelösten Textvariablen und deren Herkunftsebene anzuzeigen.
- Ergänzen oder überschreiben Sie die gewünschten Schlüssel entweder direkt in einer projektlokalen `i18n.yaml` (im Ordner neben der `markpublish.yaml`) oder in der `i18n.yaml` Ihres Themes.

B.2.4 Schriften und Schriftfamilien

Problem: Fehlermeldungen bezüglich Schriftarten oder unerwartetes Schriftbild.

Lösung:

- Das Standard-Theme nutzt die mitgelieferte Schriftfamilie **Open Sans** eine manuelle Schriftinstallation auf dem Betriebssystem ist nicht erforderlich.
- Für eigene Themes können Sie Schriftdateien (`.ttf`, `.otf`) direkt im Ordner `pdf/fonts/` Ihres Themes ablegen. markpublish bindet Theme-Schriften automatisch in den Suchpfad ein.

B.2.5 Fehlendes Metadatum (UndefinedMetadataError)

Problem: Der Build bricht mit der Meldung `Metadatum '...' wird vom Theme gelesen, ist im Dokument aber nicht gesetzt` ab.

Lösung:

- Das Theme greift per `meta.at("schluessel")` auf ein Metadatum zu, das in der `markpublish.yaml` unter `document:` nicht definiert ist und im Theme keinen Fallback besitzt.

© 2026 Frank Winter

- Tragen Sie das fehlende Feld unter `document:` in Ihrer `markpublish.yaml` ein (freie Felder sind dort uneingeschränkt zulässig), oder hinterlegen Sie im Theme einen Standardwert: `meta.at("schluessel", default: (value: none)).value`.

B.2.6 Fehlende Textbeschriftung (UndefinedLabelError)

Problem: Der Build bricht mit der Meldung `Label '...' ist im Template notiert, aber in keiner i18n-Ebene definiert` ab.

Lösung:

- Das Template greift über `labels.at("...")` auf einen Textbaustein zu, der in keiner Datei der Beschriftungskaskade für die aktive Dokumentsprache hinterlegt ist.
- Ergänzen Sie den Schlüssel unter dem entsprechenden Sprachkürzel (z. B. `de:`) in einer `i18n.yaml` direkt in Ihrem Projektordner oder im Theme.

B.2.7 Theme-Signatur passt nicht (Theme-Signaturfehler)

Problem: `markpublish labels` oder der Build melden `Theme und Aufruf passen nicht zusammen`.

Lösung:

- Die Typst-Funktion `setup-document` in `template.typ` deklariert nicht alle erforderlichen Parameter.
- markpublish übergibt alle Metadaten gesammelt über das Wörterbuch `meta: (:)`. Stellen Sie sicher, dass Ihr Theme diesen Parameter deklariert: `#let setup-document(..., meta: (:), labels: (:), body) = { ... }` (oder `..rest` akzeptiert).

B.2.8 Abweisung unbekannter Schlüssel (unknown_key)

Problem: markpublish bricht mit einer Meldung wie `chapters kennt diesen Schlüssel nicht: 'break_befor' -- meinten Sie 'break_before'?` ab.

Lösung:

- Sowohl auf oberster Ebene (`markpublish.yaml`) als auch auf Abschnitts- (`parts:`) und Kapitel-Ebene (`chapters:`) weist markpublish unbekannte Schlüssel strikt zurück, um Tippfehler frühzeitig abzufangen. Prüfen Sie den genannten Schlüssel; eigene freie Datenfelder sind ausschließlich unter `document:` zulässig.

B.2.9 Zielordner bei init ist nicht leer

Problem: `markpublish init` bricht mit der Meldung `Das Verzeichnis ... besteht bereits und ist nicht leer` ab.

Lösung:

© 2026 Frank Winter

- `init` überschreibt niemals vorhandene Dateien. Wählen Sie einen neuen Verzeichnisnamen oder leeren Sie das Verzeichnis vor der Initialisierung.

B.3 Häufig gestellte Fragen (FAQ)

B.3.1 Was ist der Unterschied zwischen Benutzeroberflächensprache und Dokumentsprache?

markpublish trennt die Sprache der Programmoberfläche strikt von der Sprache des Dokuments:

- **Programmoberfläche (`--ui-lang`, `MARKPUBLISH_UI_LANG`):** Steuert, in welcher Sprache Fehlermeldungen, Hilfetexte und Diagnoseausgaben im Terminal erscheinen (`de` oder `en`). Ohne Angabe gilt die Sprache des Betriebssystems.
- **Dokumentsprache (`language:` in `markpublish.yaml`):** Bestimmt Typografie, Silbentrennung, Datumsformat und feste Texte (wie „Inhaltsverzeichnis“ oder „Kapitel“) im PDF. Ein deutsches Terminal kann somit problemlos ein englisches PDF erzeugen.

B.3.2 Wie verhindere ich eine Trennseite vor einem Kapitel?

Standardmäßig leitet ein Kapitel mit `break_before: "page"` auf einer neuen Seite ein. Soll ein Kapitel direkt ohne Seitenwechsel anschließen, setzen Sie in der Kapiteldefinition `break_before: "none"`.

B.3.3 Warum erscheint mein Abschnitt nirgends im Dokument?

Abschnitte (`parts:`) stehen standardmäßig auf `break_before: "none"`: Sie klammern Kapitel, belegen keine eigene Seite und erscheinen nicht im Inhaltsverzeichnis. Soll der Abschnitt sichtbar werden, notieren Sie:

```yaml
parts:
  - part: "Anhänge"
    break_before: "divider"   # gestaltete Trennseite
```

Mit `break_before: "page"` erhält der Abschnitt eine Überschrift auf einer neuen Seite statt einer Trennseite.

© 2026 Frank Winter

B.3.4 Wie kann ich die Seitennummerierung für jedes Kapitel neu starten?

Setzen Sie in der `markpublish.yaml` auf Dokument- oder Kapitel-Ebene `pagenum_reset: true`. markpublish setzt die Seitenzahl zu Kapitelbeginn automatisch auf 1 zurück und berechnet die Gesamtzahl der Seiten konsistent.

B.3.5 Werden Hyperlinks im PDF klickbar exportiert?

Ja. Sowohl interne Verweise (aus dem Inhaltsverzeichnis oder Fußnoten) als auch externe Weblinks (im Markdown als `[Linktext](URL)` notiert) werden im PDF als echte, anklickbare Hyperlinks erzeugt.

© 2026 Frank Winter