

# OLLM – OSG L<sup>A</sup>T<sub>E</sub>X LECTURE MAKER

vo.15.0      23. Oktober 2026

Nutzerhandbuch

Matthias WERNER<sup>1</sup>

<https://github.com/werner-matthias/osglecture>

OLLM ist ein in Perl geschriebenes Buildscript für die `OSGLECTURE`-Klasse. Ursprünglich aus einem rc-Script für `latexmk` entstanden, baut es mit `OSGLECTURE` erstellte Vorlesungsdokumente, wählt Dokumenttyp und Sprache, startet die eigentliche Übersetzung über `latexmk`, verwaltet kapitelübergreifende Referenzen und kopiert fertige PDFs auf konfigurierte Deploymentziele.

Dieses Handbuch beschreibt die Bedienung von `OLLM` sowie die Formate der genutzten Konfigurationsdateien.

## Inhaltsverzeichnis

|                                                  |                                                           |          |
|--------------------------------------------------|-----------------------------------------------------------|----------|
| <b>1 TL;DR</b>                                   | <b>1 6 Report, Check und Doctor</b>                       | <b>7</b> |
|                                                  | 6.1 Report . . . . .                                      | 7        |
| <b>2 Schnelleinstieg</b>                         | 6.2 Check . . . . .                                       | 7        |
| 2.1 Installation . . . . .                       | 6.3 Doctor . . . . .                                      | 8        |
| <b>3 Projektaufbau</b>                           | <b>3 7 Aufräumen</b>                                      | <b>8</b> |
| 3.1 Projektwurzel und Units . . .                | 7.1 Clean . . . . .                                       | 8        |
| 3.2 Gemeinsame TeX-Konfiguration                 | 7.2 Prune . . . . .                                       | 8        |
| <b>4 Aufruf und Optionsmodell</b>                | <b>5 8 Deployment</b>                                     | <b>9</b> |
| 4.1 Allgemeine Optionen . . . . .                |                                                           |          |
| <b>5 Bauen</b>                                   | <b>9 Das Projektmanifest <code>ollmconfig.toml</code></b> | <b>9</b> |
| 5.1 Einzelner Build . . . . .                    | 9.1 Vollständiges Beispiel . . . . .                      | 9        |
| 5.2 Buildmatrix mit <code>--all</code> . . . . . | 9.2 Wurzelschlüssel . . . . .                             | 11       |
| 5.3 Abhängigkeiten mit <code>--resolve</code>    | 9.3 <code>[project]</code> und <code>[project.tex]</code> | 11       |
| 5.4 Standalone-Modus . . . . .                   | 9.4 <code>[languages]</code> . . . . .                    | 11       |
|                                                  | 9.5 <code>[targets.NAME]</code> . . . . .                 | 11       |

<sup>1</sup> [matthias.werner@informatik.tu-chemnitz.de](mailto:matthias.werner@informatik.tu-chemnitz.de)

|           |                                    |    |                                       |           |
|-----------|------------------------------------|----|---------------------------------------|-----------|
| 9.6       | [security] . . . . .               | 12 | <b>12 Definitionsdateien</b>          | <b>14</b> |
| 9.7       | [build.resolve] . . . . .          | 12 | 12.1 Bundle-Preset . . . . .          | 14        |
| 9.8       | Deploymenttabellen . . . . .       | 12 | 12.2 Targetdefinition . . . . .       | 14        |
|           |                                    |    | <b>13 Migration aus ollmconfig.pl</b> | <b>15</b> |
| <b>10</b> | <b>Lokale Projektkonfiguration</b> |    | <b>14 Exitcodes und CI</b>            | <b>15</b> |
|           | .ollmconfig.local.toml             | 13 | <b>15 Weiterführende Dokumente</b>    | <b>15</b> |
| <b>11</b> | <b>Persönliche Konfiguration</b>   | 13 | <b>Index</b>                          | <b>17</b> |

## 1 TL;DR

- Wenn Sie in einem bestehenden [OSGLECTURE](#)-Kurs<sup>1</sup> die Vorlesungsfolien einer Lehreinheit in deutscher Sprache neu generieren lassen wollen, wechseln sie zum Verzeichnis dieser Einheit und geben ein:

```
> ollm slides --language=de
```

- Sie wollen alle Scriptkapitel der Erstsprache der Lehrveranstaltung zu einem Script zusammenstellen. Dazu muss das Arbeitsverzeichnis irgendein Projektverzeichnis sein und Sie geben ein:

```
> ollm script --scope=collection
```

- Sie wollen alle Handouts der Erstsprache übersetzen. Dazu muss das Arbeitsverzeichnis irgendein Projektverzeichnis sein und Sie geben ein:

```
> ollm handout --scope=series
```

- Sie wollen alle englischen Handouts an ihren Zielort kopieren. Dazu muss das Arbeitsverzeichnis irgendein Projektverzeichnis sein und Sie geben ein:

```
> ollm deploy handout lang=en --scope=series
```

## 2 Schnelleinstieg

### 2.1 Installation

Die Installation erfolgt über mit der Installation des [OSGLECTURE](#)-Pakets über l3build:

```
> l3build install
```

---

1. D. h., mit korrekter Verzeichnisstruktur und korrektem Manifest

Dabei landet das Skript in `TEXMFHOME/scripts/osglecture/`, das nicht im Suchpfad der Shell liegt. Unter Linux und macOS genügt ein einmaliger Link in ein Verzeichnis des Suchpfads:

```
> ln -s "$(kpsewhich -var-value TEXMFHOME)/scripts/osglecture/ollm" ~/.local/bin/ollm
```

Unter Windows nehmen Sie das Verzeichnis in `PATH` auf; es enthält den Starter `ollm.cmd`.

**OLL**M benötigt für seine Ausführung Perl 5.30 oder neuer, `latexmk` und `LuaLaTeX`. Eine vollständige Installation prüfen Sie mit:

```
> ollm doctor
```

Wechseln Sie zum Anlegen eines Projekts in dessen künftige Wurzel und führen Sie aus:

```
ollm newproject --profiles=taggable
```

Der Befehl erzeugt `ollmconfig.toml`. Die Option `--profiles` ist Pflicht; ohne sie erklärt `newproject` beide Varianten und schreibt nichts:

**taggable** slides, handout und screen nutzen `ltx-talk`, script nutzt `book`. Alle Targets laufen mit `\DocumentMetadata` und eingeschaltetem Tagging; `Include/documentmetadata.tex` wird angelegt. Das setzt ein aktuelles `TeX Live` voraus, und Beamer-Themes sowie beamer-spezifischer Code funktionieren nicht.

**classic** slides, handout und screen nutzen `beamer`, script nutzt `scrbook`. Es entsteht kein Tagged PDF; davon abhängige Funktionen wie das Handout-Layout `ruled` und die Presenter-Uhr fehlen.

Die Wahl steht danach in `[targets.defaults]`; einzelne Targets lassen sich dort mit `profile = . . .` von Hand mischen. Existiert bereits eine alte `ollmconfig.pl`, wird diese soweit möglich konvertiert. Ein vorhandenes TOML-Manifest wird niemals überschrieben.

Eine kleine Serie kann so aufgebaut sein:

```
my-course/  
  ollmconfig.toml  
  Include/  
    projectconfig.tex  
    documentmetadata.tex  
  010-introduction/  
    main.tex  
  020-processes/  
    main.tex
```

In einem Unit-Verzeichnis bauen Sie beispielsweise ein deutsches Skript:

```
cd 010-introduction  
ollm build script --language=de
```

Die Kurzform `ollm script --language=de` ist gleichwertig. Die Targets `slides`, `handout`, `screen` und `script` sind mitgeliefert. `screen` erzeugt die Vortragendenfassung der Folien: Jede Seite ist doppelt so breit und trägt rechts die Presenter-Seite der Klasse `osglecture`; das Target setzt ein Profil voraus, das den Dokumenttyp `screen` unterstützt (mitgeliefert: `beamer` und `ltx-talk`). `ollm newproject` konfiguriert es wie `slides` und `handout` direkt mit.

## 3 Projektaufbau

### 3.1 Projektwurzel und Units

Die Datei `ollmconfig.toml` markiert die Projektwurzel. OLLM sucht vom Arbeitsverzeichnis aus aufwärts nach dieser Datei. Ein Unit-Verzeichnis trägt einen dreistelligen physischen Ordnungspräfix:

```
010-introduction
020-processes
```

Nach den drei Ziffern kann ein Target-Scope stehen. Die mitgelieferten Definitionen verwenden unter anderem `a/as` für Langformen und `b/bs/bh` für Präsentation, Folien und Handout. Eine optionale Rolle folgt nach einem weiteren Bindestrich:

```
030-e-scheduling-details    Excursus
090-a-appendix              Appendix
099-i-collection            Integration
```

Ohne Rollenpräfix ist die Unit regulärer Inhalt. Ein Excursus gehört zur vorhergehenden sichtbaren Content-Unit. Seine logische Ordnungskennung lautet beispielsweise `2e1`; weitere Exkurse erhalten `2e2` usw. Anhänge führen den regulären Zähler weiter und ergänzen eine lokale Kennung, beispielsweise `3ap1`, `4ap2`. Eine Integrationsunit besitzt keine Ordnungsnummer.

Die logische Unit-ID stammt nicht aus dem Verzeichnisnamen. Eine reguläre Unit deklariert sie ausdrücklich in LaTeX, beispielsweise:

```
\lecture[Prozesse]{Prozesse und Scheduling}{processes}
```

Nach einem ersten erfolgreichen Build kennt OLLM die Zuordnung von `processes` zum physischen Verzeichnis. Bis dahin ergeben unbekannte kapitelübergreifende Referenzen wie bei LaTeX üblich einen Platzhalter und eine Warnung.

### 3.2 Gemeinsame TeX-Konfiguration

Standardmäßig liegt projektweit geteiltes TeX-Material in `Include`. `Include/projectconfig.tex` enthält fachliche und darstellerische TeX-Konfiguration, beispielsweise Titel, Autor und Sprachabbildung:

```
\title{Betriebssysteme}
\author<presentation>[M. Mustermann]{Max Mustermann}
\author<longform>[Mustermann]{Max Mustermann}

\usepackage[
  languages={de,en},
  targetlang={\OsgLectureLanguage},
  map={de=ngerman,en=british}
]{langselect}
```

Titel und Sprachabbildung gehören bewusst nicht in TOML. Dadurch kann dieselbe TeX-Konfiguration auch von einem Standalone-Dokument verwendet werden.

Benötigt ein Target frühe PDF-Metadaten, enthält `Include/documentmetadata.tex` den Aufruf von `\DocumentMetadata`. Ein einfaches Beispiel ist:

```
\DocumentMetadata{lang=\OsgLectureRequestedLanguage}
```

Ob diese Datei erforderlich oder deaktiviert ist, bestimmt das jeweilige Target beziehungsweise ein Override im Projektmanifest.

## 4 Aufruf und Optionsmodell

Die allgemeine Form lautet:

```
ollm [globale Optionen] [Aktion] [Target] [Aktionsoptionen]
```

Ohne Aktion wird `build` angenommen. Mitgelieferte Targetnamen können als nacktes Wort oder mit historischem Pluspräfix geschrieben werden:

```
ollm slides
ollm +slides
ollm build --target=slides
```

Aliase sind `beamer` und `presentation` für `slides` sowie `article` für `script`. Projektdefinierte Targets werden mit `--target=NAME` gewählt.

### 4.1 Allgemeine Optionen

**--help** zeigt die eingebaute Kurzhilfe.

**--version** zeigt die OLLM-Version.

**--format=text|json** wählt menschen- oder maschinenlesbare Ausgabe. JSON eignet sich für CI und Werkzeuge.

**--color=auto|always|never** steuert farbige Terminalausgabe.

- config=DATEI** verwendet ein bestimmtes Projektmanifest.
- project-root=VERZEICHNIS** verwendet die angegebene Projektwurzel.
- non-interactive** unterdrückt interaktive Rückfragen.
- overwrite** erlaubt bei deploy das Ersetzen einer abweichenden Zieldatei, wenn die Projektpolicy eine explizite Freigabe verlangt.
- permissions=all|KEY=WERT,...** ersetzt bei deploy die konfigurierten PDF-Rechte aller Dokumenttypen, etwa `--permissions=print=none`; `all` deployt ohne Einschränkungen.
- owner-password=KENNWORT** legt bei deploy das Kennwort fest, mit dem sich die PDF-Einschränkungen aufheben lassen; ohne Angabe ist es zufällig und wird nicht gespeichert.
- +enforce+**, **--enforce+** verlangt das Pluspräfix bei Aktions- und eingebauten Targetwörtern. Das verhindert, dass ein Dateiname wie `slides` als Target interpretiert wird. `+force+` ist ein veralteter Alias.

Unbekannte kompatible Minusoptionen werden bei einem Build an `latexmk` weitergereicht. Zusätzliche RC-Dateien und Optionen, die den kontrollierten Buildvertrag umgehen würden, lehnt OLLM ab.

## 5 Bauen

### 5.1 Einzelner Build

```
ollm build script --language=en
ollm build --target=slides --language=de --source=main.tex
```

`--language` wählt eine für das Target konfigurierte Sprache. Ohne Angabe gilt die Projektstandardsprache, sofern das Target sie unterstützt, ansonsten die erste Sprache des Targets. `--source` ist standardmäßig `main.tex`.

`--scope` bestimmt, auf welche Units und Dokumentinstanzen der Build wirkt:

**current** ist der Default und baut genau das ausdrücklich gewählte beziehungsweise das Default-Target in der gewählten oder Defaultsprache der aktuellen Unit.

**unit** baut alle für die aktuelle Unit zulässigen Targets und Sprachen. Ein ausdrücklich angegebenes Target oder eine ausdrücklich angegebene Sprache schränkt diese Matrix ein.

**series** baut das gewählte beziehungsweise das Default-Target in der gewählten oder Defaultsprache für alle Units, auf die das Target anwendbar ist. Integrationsunits gehören nicht zu dieser Auswahl.

**collection** sucht für den Dokumenttyp des Targets die eindeutig passende Integrationsunit und baut dort. Der Scope funktioniert unabhängig vom Arbeitsverzeichnis. Keine oder mehrere passende Integrationsunits sind ein Fehler.

`current` und `unit` verlangen ein Unit-Verzeichnis als Ausgangspunkt; `series` und `collection` können auch aus der Projektwurzel verwendet werden.

`--rebuild` verlangt einen vollständigen `latexmk`-Neubau. `--dry-run` startet kein LaTeX, sondern zeigt die normalisierte Auswahl:

```
ollm build script --language=en --dry-run
ollm build script --language=en --dry-run --format=json
```

`--debug=tex`, `ollm` oder `tex+ollm` aktiviert Diagnoseausgaben für die gewählte Ebene. `--warnings=all`, `important` oder `none` wählt die gewünschte Warnungsmenge.

## 5.2 Buildmatrix mit `--all`

`--all` erweitert die Target-/Sprachauswahl des gewählten Build-Scopes auf alle im Manifest vorgesehenen Kombinationen. Target-Scope und Rolle jeder Unit filtern die Matrix. Ohne ausdrücklichen Scope bleibt `current` wirksam; der Befehl baut dann nicht automatisch jede Unit der Serie.

```
ollm build --all
```

## 5.3 Abhängigkeiten mit `--resolve`

`--resolve` verfolgt die von LaTeX gemeldeten serieninternen Abhängigkeiten und baut veraltete Produzenten beziehungsweise Konsumenten bis zu einem Fixpunkt:

```
ollm build script --resolve
```

Zyklen sind erlaubt. Noch nie direkt gebaute logische Units kann OLLM nicht aus Verzeichnisnamen erraten; bauen Sie eine solche Unit zunächst einmal in der benötigten Kombination aus Dokumenttyp und Sprache. Die Höchstzahl der Fixpunktrunden wird in `[build.resolve]` konfiguriert.

## 5.4 Standalone-Modus

Für ein Dokument ohne Serienmanifest bleibt die kompatible Aktion `standalone` verfügbar:

```
ollm standalone main.tex
```

Dokumenttyp und Sprache stammen dabei aus den Klassenoptionen des Dokuments; `--target`, `--language` und `--all` sind nicht zulässig.

## 6 Report, Check und Doctor

### 6.1 Report

report beschreibt Units, vorhandene Dokumentinstanzen, Ausgabedateien, den jeweils gültigen Buildstand, Kapitel- und Ordnungsnummern, Konfigurationssignaturen und Abhängigkeiten. Befunde ändern seinen Exitstatus nicht, solange der Zustand lesbar ist.

```
ollm report
ollm report --scope=unit
ollm report --format=json
```

In der Projektwurzel ist der Default-Scope series; in einer Unit current. unit muss ausdrücklich gewählt werden. Eine entdeckte, aber unbenutzte und ungebaute Unit erscheint als dormant und ist kein Fehler.

### 6.2 Check

check liest dieselben Daten, bewertet Inkonsistenzen jedoch als Fehler. Es baut nichts. Geprüft werden unter anderem Artefakte, Identitäten, Rollen und Scopes, Target-/Sprachzulässigkeit, Job-ID, Ordnungsnummer und Referenzabhängigkeiten.

```
ollm check
ollm check --scope=current --target=script --language=de
ollm check --format=json
```

Ein inkonsistenter erforderlicher Zustand ergibt Exitcode 3. Ruhende Units erzeugen keine Buildpflicht. Die aggregierte Konfiguration kann sich bereits durch eine neue ruhende Unit ändern; ein solcher unspezifischer Unterschied wird daher als Warnung gemeldet, konkrete Vertragsabweichungen bleiben Fehler.

### 6.3 Doctor

doctor prüft global Perl, latexmk, LuaLaTeX, kpsewhich, den TOML-Parser sowie erforderliche und optionale TeX-Pakete. Innerhalb eines Projekts prüft es zusätzlich Manifest, Definitionen, Projektdateien, Schreibbarkeit, gespeicherten Zustand und Shell-Escape-Policy.

```
ollm doctor
ollm doctor --project-root=/path/to/project
ollm doctor --format=json
```

Fehlende notwendige Werkzeuge ergeben Exitcode 69. Optionale Pakete werden angezeigt, führen allein aber nicht zum Fehlschlag.

## 7 Aufräumen

### 7.1 Clean

`clean` entfernt ausgewählte Arbeits- oder Zustandsdaten. Der Default-Level ist `aux`, der Default-Scope `current`.

`--level=aux` entfernt Hilfsdateien, behält aber das lokale PDF.

`--level=build` entfernt das isolierte Buildverzeichnis.

`--level=state` entfernt den gespeicherten Projektzustand.

`--level=all` kombiniert `build` und `state`.

Scopes sind `current`, `unit` und `series`. Ein Serien-Clean aus einer Unit verlangt eine Rückfrage; von der Projektwurzel aus nicht. Mit `--dry-run` wird nur der Löschanplan gezeigt.

```
ollm clean --level=aux
ollm clean --level=build --scope=unit --dry-run
ollm clean --level=state --scope=series
```

### 7.2 Prune

`prune` entfernt aufgegebene temporäre Zustände und ersetzt ältere Buildstände. Zustände von inzwischen fehlenden physischen Units werden nur mit `--stale-units` entfernt:

```
ollm prune --dry-run
ollm prune --stale-units
```

Wenn kein OLLM- oder `latexmk`-Prozess läuft, darf das gesamte Verzeichnis `.osglecture` als radikale Reparatur gelöscht werden. Dabei gehen auch alle bekannten Zuordnungen logischer Unit-IDs verloren; betroffene Units müssen anschließend wieder direkt gebaut werden.

## 8 Deployment

`deploy` kopiert bereits erfolgreich gebaute PDFs. Es startet keinen Build und legt keine Zielverzeichnisse an.

```
ollm deploy handout
ollm deploy --target=script --scope=series
ollm deploy --target=script --scope=collection
ollm deploy --all --overwrite
```

In der Projektwurzel gilt standardmäßig `series`, in einer Unit `current.unit` wählt alle vorhandenen Dokumentinstanzen der Unit; `collection` wählt die Integrationsdatei des Dokumenttyps unabhängig vom Arbeitsverzeichnis. Bei Serien-Deployment entscheidet `deployment.series`, ob Einzelunits, Collection oder beide kopiert werden.

Fehlt ein Zielpfad, meldet OLLM einen Fehler und überspringt weitere Versuche mit demselben Pfad. Andere konfigurierte Ziele werden weiter bearbeitet. Eine vorhandene identische Datei bleibt unverändert. Ob eine abweichende Datei `--overwrite` verlangt, bestimmt `security.deployment.overwrite`.

Deployte Kopien können eingeschränkte PDF-Rechte tragen, etwa keine Druckerlaubnis. `[deployment.permissions]` ist der Default aller Dokumenttypen, `[deployment.types.DOCTYPE.permissions]` ersetzt ihn vollständig; nicht genannte Rechte bleiben erlaubt:

```
[deployment.permissions]
print = "none"      # full | low | none
extract = false     # Text und Grafik kopieren
modify = false      # Seiteninhalt ändern
annotate = false    # kommentieren
form = false        # Formulare ausfüllen und signieren
assemble = false    # Seiten einfügen, löschen, drehen
```

Eingeschränkte Kopien erzeugt OLLM mit dem externen Programm `qpdf` (AES-256, ohne Öffnungskennwort, mit zufälligem oder per `--owner-password` angegebenem Eigentümerkennwort). Fehlt `qpdf`, fragt `deploy` im Terminal, ob ohne Einschränkungen kopiert werden soll; mit `--non-interactive` oder ohne Terminal bricht es mit Exitcode 69 ab. Die Einschränkungen sind leicht zu entfernen und daher ein Hinweis an die Lesenden, keine Zugriffskontrolle.

## 9 Das Projektmanifest `ollmconfig.toml`

### 9.1 Vollständiges Beispiel

```
schema = 1
bundle_preset = "OSG lecture/1"

[project]
id = "bs"

[project.tex]
directory = "Include"
config = "projectconfig.tex"

[languages]
available = ["de", "en"]
default = "de"
```

```
[targets.slides]
languages = ["de", "en"]
document_metadata = "disabled"

[targets.handout]
languages = ["de"]

[targets.script]
languages = ["de", "en"]
document_metadata = "required"

[security]
shell_escape = "restricted"

[build.resolve]
max_rounds = 8

[deployment]
series = "both"

[deployment.roles]
content = ""
appendix = "A"
excursus = "E"
integration = ""

[deployment.types.handout]
paths = ["deployment/handouts", "/srv/lecture/handouts"]
filename = "{role}{chapter:02}-{unit}-{lang}.pdf"
collection_filename = "{series}-{doctype}-{lang}.pdf"
series = "units"

[deployment.types.handout.units.introduction]
filename = "{chapter:02}-intro-{lang}.pdf"

[security.deployment]
overwrite = "explicit"
```

Unbekannte Schlüssel sind Fehler. Alle Bezeichner und Pfade müssen die in den folgenden Abschnitten beschriebenen Bedingungen erfüllen.

## 9.2 Wurzelschlüssel

**schema** Pflichtwert 1. Er bezeichnet das Dateischema, nicht die OLLM- oder Bundleversion.

**bundle\_preset** Referenz auf ein Bundle-Preset, gewöhnlich "OSG lecture/1". Fehlt der Schlüssel, verwendet OLLM den Nutzerdefault beziehungsweise den eingebauten Default.

Weitere erlaubte Tabellen sind `project`, `languages`, `targets`, `security`, `build` und `optional deployment`.

### 9.3 [project] und [project.tex]

`project.id` ist die technische, portable Serien-ID. Sie geht in Jobnamen und Dokumentidentitäten ein und sollte nach dem ersten Build nicht leichtfertig geändert werden.

`project.tex.directory` ist ein projektwurzelrelatives Verzeichnis; Default ist `Include`. Absolute Pfade und Pfade oberhalb der Projektwurzel sind nicht zulässig. `project.tex.config` ist ein reiner Dateiname ohne Verzeichniskomponenten; Default ist `projectconfig.tex`. Die Datei darf neben Einstellungen auch projektlokale Pakete aus dem gemeinsamen Verzeichnis verwenden.

### 9.4 [languages]

`languages.available` ist eine nichtleere Liste portabler kurzer Sprachkennungen. Doppelte Werte und Kollisionen, die sich nur in Groß- und Kleinschreibung unterscheiden, sind unzulässig. `languages.default` muss in dieser Liste stehen.

Diese Tabelle beschreibt ausschließlich die Buildmatrix. Die Zuordnung zu `babel`- oder anderen LaTeX-Sprachnamen gehört in `projectconfig.tex`.

### 9.5 [targets.NAME]

Für jedes Projekt-Target ist eine Tabelle erforderlich. `languages` ist eine nichtleere Teilmenge von `languages.available`. Optional überschreibt `document_metadata = "required"` oder `"disabled"` den Default der Targetdefinition.

Der Name muss durch eine eingebaute oder zusätzlich gefundene Targetdefinition registriert sein. Schema 1 verlangt, dass Targetname und Doctype identisch sind.

### 9.6 [security]

`shell_escape` kann folgende Werte annehmen:

**off** startet LuaLaTeX mit deaktiviertem Shell-Escape.

**restricted** verwendet die eingeschränkte Allowlist der TeX-Distribution; dies ist der Default.

**full** erlaubt vollständiges Shell-Escape und sollte nur für vertrauenswürdige Quellen verwendet werden.

In `[security.deployment]` bestimmt `overwrite` das Überschreiben abweichender Zieldateien:

**explicit** verlangt bei Bedarf `--overwrite`; Default.

**automatic** ersetzt abweichende Dateien ohne Zusatzoption.

## 9.7 [build.resolve]

`max_rounds` ist eine positive Ganzzahl und begrenzt den Fixpunktlauf von `build --resolve`. Ohne Angabe gelten acht Runden.

## 9.8 Deploymenttabellen

`deployment.series` legt die Serienauswahl global fest: "units", "collection" oder "both". Ohne Angabe gilt "both". In `deployment.roles` können die Rollen `content`, `appendix`, `excursus` und `integration` auf frei wählbare Dateinamensbestandteile abgebildet werden.

Für jeden deploybaren Dokumenttyp ist `[deployment.types.DOCTYPE]` erforderlich:

**paths** nichtleere Liste von Zielverzeichnissen. Relative Pfade beziehen sich auf die Projektwurzel. OLLM legt sie nicht an.

**filename** Schablone für einzelne Units.

**collection\_filename** optionale Schablone für die Integrationsdatei; ohne sie wird `filename` verwendet.

**series** optionaler Override von `deployment.series` für diesen Dokumenttyp.

Ein Dateiname für eine einzelne logische Unit kann ausschließlich über eine Unit-Tabelle überschrieben werden:

```
[deployment.types.DOCTYPE.units.UNIT-ID]
filename = "..."
```

Schablonen kennen:

**{series}** technische Projekt-ID;

**{unit}** logische Unit-ID;

**{slug}** beschreibender Teil des Unit-Verzeichnisnamens ohne Nummer, Scope-Code und Rollenkürzel, etwa `processes` für `030-processes` oder `010a-e-processes`;

**{ordinal}** logische Ordnungskennung;

**{chapter}** von LaTeX gemeldete Kapitelrepräsentation;

**{doctype}** Dokumenttyp;

**{lang}** Sprache;

**{role}** Wert aus `deployment.roles`.

Dezimale Werte können mit einer Mindestbreite versehen werden, etwa `{chapter:02}`. Bei `ordinal` wird nur der numerische Präfix aufgefüllt: `1e2` wird zu `01e2`, `3ap1` zu `03ap1`. Die Kapitelrepräsentation kann im Dokument mit `\0sgLectureDeploymentChapter{...}` ausdrücklich gesetzt werden.

## 10 Lokale Projektkonfiguration `.ollmconfig.local.toml`

Diese optionale, üblicherweise nicht versionierte Datei erweitert ausschließlich die Suchpfade für Definitionen:

```
schema = 1

[definitions]
paths = ["Definitions", "/opt/osglecture/definitions"]
```

`schema` muss 1 sein. `definitions.paths` ist eine Liste; relative Einträge beziehen sich auf die Projektwurzel. Alle Pfade müssen existieren. Doppelte Definitionen desselben Namens sind ein Fehler. Die lokalen Pfade werden vor den mitgelieferten Definitionen durchsucht, ersetzen aber nicht stillschweigend eine gleichnamige Definition.

## 11 Persönliche Konfiguration

Die persönliche Konfiguration enthält gegenwärtig nur den bevorzugten Bundle-Preset:

```
schema = 1
bundle_preset = "OSG lecture/1"
```

OLLM sucht sie unter Unix zunächst in `$XDG_CONFIG_HOME/ollm/config.toml` und sonst in `$HOME/.config/ollm/config.toml`, unter Windows in `%APPDATA%\ollm\config.toml`. Die Umgebungsvariable `OLLM_USER_CONFIG` kann eine andere Datei bestimmen. Ein expliziter Wert im Projektmanifest hat Vorrang.

## 12 Definitionsdateien

Definitionsdateien sind TOML-Dateien in den mitgelieferten oder lokal konfigurierten Suchpfaden. Normale Projektanwender müssen sie nur für eigene Targets bearbeiten.

### 12.1 Bundle-Preset

```
schema = 1
kind = "bundle-preset"
name = "OSG lecture"
version = "1"
```

Erlaubt sind ausschließlich `schema`, `kind`, `name` und `version`. Das Projekt referenziert das Preset als `"Name/Version"`, hier also `"OSG lecture/1"`.

## 12.2 Targetdefinition

```

schema = 1
kind = "target"
name = "studyguide"
version = "1.0"
doctype = "studyguide"
profile_class = "longform"
document_metadata = "required"
unit_scopes = ["a", "as"]

```

**name** registrierter CLI-Targetname.

**version** Version dieser Definition, unabhängig vom Schema.

**doctype** Dokumenttyp; in Schema 1 identisch zu name.

**profile\_class** "presentation" oder "longform". Die konkrete TeX-Profilwahl erfolgt in `projectconfig.tex`.

**document\_metadata** Default "required" oder "disabled"; das Projekt darf ihn überschreiben.

**unit\_scopes** optionale Liste ein- oder zweibuchstabiger Verzeichnisscopes, für die das Target gilt.

Legen Sie eine eigene Definition beispielsweise hier ab:

Definitions/targets/studyguide.toml

Tragen Sie anschließend Definitionen in die lokale Projektkonfiguration ein, ergänzen Sie `[targets.studyguide]` im Projektmanifest und wählen Sie das Target mit `--target=studyguide`.

## 13 Migration aus ollmconfig.pl

```

ollm convertproject
ollm newproject --profiles=classic

```

`convertproject` sucht eine alte `ollmconfig.pl`, erzeugt eine konservative `ollmconfig.toml`, ohne den Perl-Code auszuführen, und übernimmt erkennbare Metadaten aus `Include/lectdates.tex` nach `Include/projectconfig.tex`. `newproject` erzeugt Manifest, Include-Verzeichnis und eine Projektkonfiguration mit Dummy-Metadaten oder verhält sich bei gefundenen Legacy-Dateien wie `convertproject`. `convertproject` verwendet ohne `--profiles` den Profilsatz `classic`, also die Klassen des alten Builds. Bestehende Konfigurationsdateien werden nicht überschrieben.

Statisch erkennbare Sprachen, Shell-Escape-Einstellung, gemeinsames TeX-Verzeichnis, einfache Deploymentzuweisungen und die von `osglecture` unterstützten Metadatenbefehle aus `lectdates.tex` werden übernommen. Dabei wird `\ldeenr` zu `\ldeen` vereinheitlicht, ein `\input` zu `\IncludeOsgLecturePreamble` (das Fragment läuft dann *nach* `\LoadClass`), und die Sprachreihenfolge des Manifests wird als `\LectureProjectSetup` mit `selectable`-Liste in `projectconfig.tex`

gespiegelt. Altbefehle ohne `osglecture`-Entsprechung (`\tucurl`, `\logo`) werden auskommentiert übernommen. Dynamische Perl-Ausdrücke, Passwörter, Deploymentrestriktionen, gemeinsam genutzte Datenpfade und alte Kapitelnummernlogik müssen anhand der ausgegebenen Warnungen manuell übertragen werden. Eine automatisch erzeugte `projectconfig.tex` sollte früh einmal testweise übersetzt werden.

Während der Migration dürfen beide Konfigurationsdateien nebeneinander liegen; gebaut wird ausschließlich mit TOML. Liegt nur `ollmconfig.pl` vor, bricht **OLLM** mit einem Hinweis auf `convertproject` ab.

## 14 Exitcodes und CI

- 0** erfolgreiche Aktion beziehungsweise lesbarer Report;
- 1** Build-, Deployment- oder Zustandsfehler;
- 2** ungültiger Aufruf oder ungültige Konfiguration;
- 3** check hat eine Inkonsistenz gefunden;
- 69** benötigtes Programm oder eine nicht verfügbare Ausführungsfunktion;
- 128+Signal** signalbedingter Abbruch auf POSIX-Systemen, insbesondere 130 bei Strg-C.

Für maschinelle Auswertung sollte `--format=json` verwendet werden. `report` ist kein Freigabegate; für CI ist `check` vorgesehen.

## 15 Weiterführende Dokumente

Dieses Handbuch beschreibt ausschließlich die Benutzerschnittstelle. Hinweise für Wartung und Weiterentwicklung stehen außerhalb der installierten Nutzerdokumentation:

**development-docs/DESIGN.md** OLLM-Architektur, Dateiverträge, Invarianten und Rationales;

**development-docs/ARCHITECTURE.md** Architektur von `osglecture`;

**development-docs/GLOSSARY.md** verbindliche Begriffe;

**development-docs/DEPENDENCIES-ollm.md** technische Herkunft und Reparaturhinweise zu Laufzeitabhängigkeiten;

**development-docs/DEVELOPMENT-ollm.md** Build-, Test- und Paketierungshinweise für Mitwirkende;

**ollm/THIRD\_PARTY.md** Lizenzen mitgelieferter Komponenten.

# Index

|   |                                             |
|---|---------------------------------------------|
| L | <a href="#">latexmk</a> (Paket) . . . . . 1 |
|---|---------------------------------------------|