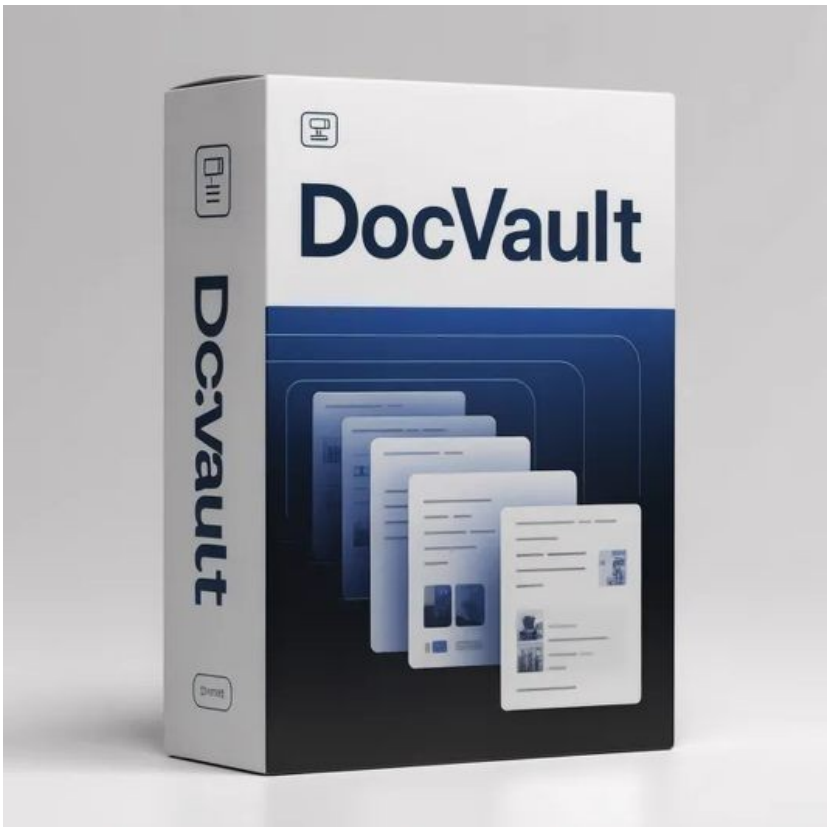
DocVault

Technische Dokumentation

Version 4.1.3

Einstellungen • Suchindex • Stoppwörter • Plugin-System

Stand: März 2026



Dual-Settings-Architektur

DocVault verwendet ein zweigeteiltes Einstellungssystem, das speziell für den Multi-PC-Betrieb über Dropbox, OneDrive oder Netzlaufwerke entwickelt wurde. Einstellungen, die PC-spezifisch sind (z.B. Scanner-Konfiguration), werden lokal gespeichert. Einstellungen, die für alle PCs gelten sollen (z.B. Kategorien), werden im gemeinsamen Datenordner abgelegt.

Eigenschaft	Beschreibung
`Lokale Settings`	Gespeichert neben der EXE/dem Skript. Gelten nur für diesen PC.
`Gemeinsame Settings`	Gespeichert im Datenordner (DATA_DIR). Gelten für alle PCs die auf denselben Datenordner zugreifen.
`Speicherformat`	Beide als JSON-Dateien (UTF-8, formatiert mit Einrückung).
`Ladepriorität`	Lokale Settings werden zuerst geladen, dann gemeinsame. Lokale haben Vorrang bei Konflikten.

Lokale Einstellungen

Speicherort

Datei: `docvault_local.json`

Pfad: Im selben Verzeichnis wie DocVault.exe (bzw. das Python-Skript)

Diese Datei wird beim ersten Start automatisch erstellt. Sie enthält alle Einstellungen, die nur für diesen einen Computer relevant sind.

Alle lokalen Schlüssel

Schlüssel	Typ / Standard	Beschreibung
`appearance`	"dark"	Darstellungsmodus: "dark" oder "light"
`sort`	"date_desc"	Aktive Sortierung der Dokumentliste (z.B. date_desc, size_asc, name_az)
`page_size`	10	Dokumente pro Seite (5, 10, 20, 50, 100)
`sidebar_width`	260	Breite der Sidebar in Pixeln (per Drag änderbar)
`preview_width`	350	Breite des Vorschau-Panels in Pixeln
`scanner_name`	""	Name des ausgewählten Scanners (WIA-Gerätename)
`naps2_path`	""	Manuell gesetzter NAPS2-Pfad (falls nicht automatisch gefunden)
`scan_profile`	""	Name des aktiven NAPS2-Profils
`scan_dpi`	"150"	Scan-Auflösung (150, 200, 300, 600)
`scan_color`	"Graustufen"	Farbmodus (Farbe, Graustufen, SW)
`scan_source`	"Einzug (ADF)"	Papierquelle (Einzug (ADF), Flachbett)
`scan_fmt`	"PDF"	NAPS2-Ausgabeformat
`scan_store_fmt`	"PDF (Standard)"	Speicherformat: PDF (Standard) oder WebP (komprimiert)
`scan_ocr_lang`	"deu"	OCR-Sprache beim Scannen
`scan_ocr_mode`	"Am besten"	OCR-Modus (Am besten, Schnell)
`scan_deskew`	false	Automatisches Geraderichten beim Scan
`enabled_plugins`	[]	Liste der aktivierten Plugin-Dateinamen (z.B. ["preview_alibre", "preview_word"])
`max_ocr_chars`	50000	Suchindex: Max. Textzeichen pro Dokument
`max_index_words`	5000	Suchindex: Max. einzigartige Wörter im Index pro Dokument
`min_word_len`	3	Suchindex: Minimale Wortlänge für den Index
`search_history`	[]	Letzte Suchbegriffe (max. 20)
`excluded_cats`	[]	Ausgeblendete Kategorien in der Seitenleiste
`tree_expanded`	[]	Aufgeklappte Kategorien im Baum

Dropbox-Konfliktauflösung

Wenn mehrere PCs über Dropbox auf denselben Ordner zugreifen, kann Dropbox Konfliktkopien der lokalen Einstellungsdatei erstellen (z.B. docvault_local (Konflikt von Max-PC).json). DocVault erkennt diese automatisch beim Start:

1. Alle Dateien mit dem Muster docvault_local*conflicted*.json werden gesucht.

2. Falls die Hauptdatei docvault_local.json fehlt, wird die neueste Konfliktkopie wiederhergestellt.

3. Alle Konfliktkopien werden gelöscht (aufgeräumt).

✅ **Tipp:** Da lokale Einstellungen PC-spezifisch sind (Scanner, Darstellung), ist es normal und unproblematisch, dass jeder PC seine eigene Version hat.

Gemeinsame Einstellungen

Speicherort

Datei: `settings.json`

Pfad: Im Datenordner (DATA_DIR), z.B. C:\DocVault\data\settings.json

Diese Einstellungen gelten für alle PCs, die auf denselben Datenordner zugreifen. Änderungen auf einem PC sind sofort auf allen anderen PCs wirksam.

Alle gemeinsamen Schlüssel

Schlüssel	Typ / Standard	Beschreibung
`ocr_lang`	"deu"	Standard-OCR-Sprache für die Verarbeitung importierter Dokumente
`link_only`	false	Standard-Import-Modus: false = Kopieren, true = Nur verknüpfen

Die gemeinsamen Einstellungen sind bewusst minimal gehalten. Kategorien und deren Schlüsselwörter werden in der SQLite-Datenbank gespeichert (Tabelle categories), nicht in der JSON-Datei.

Datenpfad-Konfiguration

Konfigurationsdatei: `docvault_pfad.ini`

Der Datenpfad kann in den Einstellungen geändert werden. Er wird in einer INI-Datei neben der EXE gespeichert:

```
[DocVault]
daten_pfad = D:\\Dropbox\\DocVault
```

Falls diese Datei nicht existiert oder leer ist, wird der Datenordner automatisch neben der EXE erstellt. Typische Szenarien:

Szenario	Datenpfad
`Einzelplatz (Standard)`	Neben der EXE (z.B. C:\DocVault\docvault_data\)
`Dropbox-Sync`	D:\Dropbox\DocVault\
`OneDrive-Sync`	C:\Users\Name\OneDrive\DocVault\
`Netzlaufwerk`	Z:\Gemeinsam\DocVault\
`USB-Stick (portabel)`	E:\DocVault\data\ (relativ zur EXE)

Suchindex & Volltextsuche

DocVault implementiert eine eigene Volltextsuche auf Basis der SQLite-Datenbank. Jedes Dokument wird beim Import oder Scan in einzelne Wörter zerlegt (Tokenisierung), gefiltert und normalisiert. Die resultierenden Terme werden in der Tabelle search_index gespeichert.

Pipeline der Textverarbeitung

Wenn ein Dokument importiert oder gescannt wird, durchläuft der Text folgende Verarbeitungsschritte:

1. Textextraktion: OCR für PDFs/Bilder (via NAPS2), Plugin-Extraktion für Office/CAD, oder direktes Lesen für Textdateien.

2. Zeichenbegrenzung: Der Rohtext wird auf max_ocr_chars Zeichen begrenzt (Standard: 50’000). Überschüssiger Text wird abgeschnitten.

3. Tokenisierung: Der Text wird in Kleinbuchstaben konvertiert und an Nicht-Wort-Zeichen aufgespalten.

4. Stoppwort-Filterung: Wörter, die in der Stoppwort-Liste stehen, werden entfernt (431 Wörter, siehe unten).

5. Längenfilter: Wörter kürzer als min_word_len werden entfernt (Standard: 3 Zeichen).

6. Normalisierung: Umlaute werden zu Basisformen aufgelöst (ä→ae, ö→oe, ü→ue, ß→ss). Beide Varianten werden gespeichert.

7. Deduplizierung: Nur einzigartige Terme werden gespeichert.

8. Begrenzung: Maximal max_index_words Terme pro Dokument (Standard: 5’000). Kürzere Wörter werden bevorzugt (sind oft spezifischer: IBAN, CHF, PDF).

Konfigurierbare Parameter

Parameter	Standard	Beschreibung
`max_ocr_chars`	50’000	Max. Zeichen Rohtext pro Dokument in der DB. 400 Seiten ≈ 400’000 Zeichen. Wählbar: 10K / 50K / 100K / 200K / 500K.
`max_index_words`	5’000	Max. einzigartige Wörter im Suchindex pro Dokument. Rechnung ≈ 50–200, Brief ≈ 100–500, Buch ≈ 5K–30K. Wählbar: 1K / 5K / 10K / 20K / 50K.
`min_word_len`	3	Minimale Wortlänge. Wörter kürzer als dieser Wert werden nicht indexiert. Wählbar: 2 / 3 / 4 Zeichen.
⚠ **Wichtig:** Änderungen an den Suchindex-Einstellungen wirken nur auf neu importierte/gescannte Dokumente. Bestehende Dokumente müssen über „OCR erneut ausführen“ neu indexiert werden.		

Stoppwörter-System

Stoppwörter sind häufige Wörter ohne eigenständigen Informationsgehalt (z.B. „der“, „die“, „und“, „the“, „and“). Diese Wörter kommen in fast jedem Dokument vor und würden den Suchindex unnötig aufblähen, ohne die Suche zu verbessern.

Funktionsweise

DocVault enthält eine fest eingebaute Liste von 431 Stoppwörtern (Deutsch + Englisch). Diese Liste wird bei der Tokenisierung als Set-Lookup verwendet – ein extrem schneller Vorgang mit O(1)-Zeitkomplexität.

```
# Tokenisierung (vereinfacht)
def tokenize(text):
    words = text.lower().split()
    return [w for w in words
            if len(w) >= min_word_len
            and w not in STOP]  # ← Stoppwort-Filter
```

Enthaltene Stoppwörter (431)

Deutsche Stoppwörter (ca. 340)

Artikel (der, die, das, den, dem, des, ein, eine ...), Pronomen (ich, du, er, sie, es, wir, ihr ...), Präpositionen (mit, von, für, auf, nach, bei, aus, über, zum, zur, bis, vor, durch, unter ...), Konjunktionen (und, oder, aber, doch, wenn, weil, dass ...), Hilfsverben (ist, sind, war, hat, haben, wird, werden, wurde, kann, soll, muss ...), Adverbien und Füllwörter (auch, noch, nur, sehr, viel, schon, immer, wieder, bereits, etwa, ziemlich ...).

Englische Stoppwörter (ca. 90)

Articles & pronouns (the, and, for, that, this, you, her, our ...), auxiliaries (was, has, have, been, will, would, could, should ...), prepositions (with, from, into, over, about, after, before, between ...), conjunctions (but, not, also, than, because, while, during ...).

Warum keine benutzerdefinierten Stoppwörter?

Die Stoppwort-Liste ist bewusst fest eingebaut und nicht konfigurierbar:

Konsistenz: Alle Dokumente werden mit denselben Regeln indexiert. Eine Änderung der Liste würde bestehende Indizes inkompatibel machen.

Geschwindigkeit: Das fest eingebaute Python-Set ist extrem schnell (O(1) pro Lookup).

Zuverlässigkeit: Die Liste wurde sorgfältig kuratiert. Zu aggressive Filterung würde relevante Wörter entfernen, zu wenig Filterung den Index aufblähen.

Synonym-Erweiterung

Zusätzlich zur Stoppwort-Filterung verfügt DocVault über ein Synonym-System. Bei der Suche werden verwandte Begriffe automatisch ergänzt:

Suchbegriff	Wird erweitert um
`arzt`	doktor, medizin, praxis, dr, ärztin
`rechnung`	invoice, zahlung, faktura
`vertrag`	vereinbarung, kontrakt
`miete`	mietvertrag, vermieter, wohnung

steuer	finanzamt, steuererklärung, elster
bank	konto, kontoauszug, iban
gehalt	lohn, lohnabrechnung, salär, einkommen
auto	fahrzeug, kfz, pkw, wagen
haus	wohnung, immobilie, gebäude

Die Synonym-Erweiterung funktioniert bidirektional: Wer nach „invoice“ sucht, findet auch Dokumente mit „Rechnung“.

Plugin-System

DocVault verfügt über ein modulares Plugin-System, das sowohl die Dokumentvorschau als auch die Volltextsuche für beliebige Dateiformate erweitert. Plugins sind einfache Python-Dateien im plugins/-Ordner.

Funktionsprinzip

1. Beim Start scannt DocVault den plugins/-Ordner nach .py-Dateien und lädt alle Module via importlib.

2. Jedes Plugin muss ein PLUGIN_INFO-Dictionary exportieren mit Name, Beschreibung, Version und unterstützten Dateiendungen.

3. Beim Klick auf ein Dokument prüft DocVault die Dateiendung gegen die Extension-Listen aller aktiven Plugins.

4. Das erste passende Plugin wird aufgerufen. Die Zuordnung basiert ausschliesslich auf der extensions-Liste (nicht auf can_preview).

5. Optional kann ein Plugin eine can_preview()-Funktion haben, die als Filter dient (z.B. Dateigrösse prüfen). Diese kann ein Plugin ablehnen, aber nicht allein matchen.

Plugin-Schnittstelle

Jedes Plugin muss folgende Elemente bereitstellen:

PLUGIN_INFO (Pflicht)

```
PLUGIN_INFO = {
    "name": "Mein Plugin",
    "description": "Beschreibung des Plugins",
    "version": "1.0",
    "author": "Autor",
    "extensions": [".xyz", ".abc"],  # Unterstützte Dateiendungen
    # Optional für Wildcard-Matching:
    "extension_prefixes": [".ad_"],  # Alle .ad_* Endungen
}
```

render_preview(filepath, max_width) – Vorschau

```
def render_preview(filepath, max_width=330):
    """Gibt ein Dictionary zurück:
    {"type": "text"|"image"|"html_text"|"error",
     "content": str,  # Anzuzeigender Text/HTML
     "pages": int,    # Seitenanzahl
     "info": str}     # Einzeiler für Statuszeile
    """
```

extract_text(filepath) – Volltextsuche

```
def extract_text(filepath):
    """Gibt (text, page_count) zurück.
    Der Text wird tokenisiert und im Suchindex gespeichert."""
    return "extrahierter text...", 1
```

can_preview(filepath) – Filter (optional)

```
def can_preview(filepath):
    """Kann das Plugin die Datei verarbeiten?
```

```
    Dient nur als Filter (z.B. Dateigrösse).
    KANN ein Plugin ABLEHNEN, aber NICHT allein matchen."""
    return os.path.getsize(filepath) < 100_000_000
```

Zuordnungs-Logik (Matching)

Die Zuordnung einer Datei zu einem Plugin folgt einem strikten Regelwerk:

Schritt 1: Prüfe ob die Dateiendung in der extensions-Liste des Plugins vorkommt (exakter Match).

Schritt 2: Falls nicht, prüfe ob die Endung mit einem Eintrag in extension_prefixes beginnt (Prefix-Match, z.B. .ad_ für alle Alibre-Dateien).

Schritt 3: Falls weder Extension noch Prefix passt → Plugin wird übersprungen.

Schritt 4: Falls can_preview() existiert und False zurückgibt → Plugin wird übersprungen (z.B. Datei zu gross).

Schritt 5: Erstes passendes Plugin gewinnt. Reihenfolge: alphabetisch nach Dateiname.

⚠ **Wichtig:** can_preview() allein kann ein Plugin NICHT triggern. Es dient ausschliesslich als Ablehnungs-Filter. Dies verhindert, dass ein generisches Plugin (z.B. preview_text.py) alle Dateien an sich zieht.

Mitgelieferte Plugins

Plugin-Datei	Version	Beschreibung
preview_alibre.py	v3.0	Alibre Design CAD-Dateien. Extrahiert Core Properties (24 GUID-basierte Felder) und Custom Properties aus dem STEP_DATA-Stream via STRINGVALUEPAIR-Parsing. Löst Material-GUIDs auf, konvertiert Zeitstempel und Sonderzeichen.
preview_binary.py	v1.0	Binärformate: Excel (openpyxl + ZIP-XML Fallback), PowerPoint (ZIP-XML), E-Mail (.eml), Archive (.zip/.tar), RTF (Regex-Parser), altes Word (.doc), STEP- und IGES-CAD-Dateien.
preview_word.py	v1.0	Modernes Word (.docx). Benötigt python-docx. Extrahiert Text, Tabellen und Core-Properties (Autor, Titel etc.).
preview_libreoffice.py	v1.0	OpenDocument-Formate (.odt, .ods, .odp, .odg). Liest direkt aus dem ZIP-Archiv – nur Standardbibliothek, kein LibreOffice nötig.
preview_html.py	v1.0	HTML-Dateien (.html, .htm, .xhtml). Bereinigt Tags via html.parser mit Regex-Fallback falls html.parser im PyInstaller-Build fehlt.
preview_text.py	v1.0	32+ Textformate (.txt, .csv, .md, .json, .py, .js, .sql, .xml, .yaml, .ini, .log etc.). Encoding-Erkennung mit UTF-8/CP1252/Latin-1 Fallback-Kette.

Eigene Plugins erstellen

Um ein eigenes Plugin zu erstellen, legen Sie eine .py-Datei im plugins/-Ordner an. Minimales Beispiel:

```
"""DocVault Plugin: Mein Format"""

PLUGIN_INFO = {
    "name": "Mein Format",
    "description": "Liest .xyz-Dateien",
    "version": "1.0",
    "extensions": [".xyz"],
}

def render_preview(filepath, max_width=330):
```

```
    with open(filepath, "r") as f:
        text = f.read()[:5000]
    return {"type": "text", "content": text,
            "pages": 1, "info": "XYZ-Datei"}

def extract_text(filepath):
    with open(filepath, "r") as f:
        return f.read()[:50000], 1
```

Aktivieren Sie das Plugin in den Einstellungen oder klicken Sie auf „Plugins neu laden“.

Datenbank-Schema

DocVault verwendet SQLite 3 mit WAL-Modus (Write-Ahead Logging) für gleichzeitigen Lese-/Schreibzugriff und einem Busy-Timeout von 30 Sekunden. Beim Beenden wird automatisch VACUUM ausgeführt.

Tabelle: documents

Spalte	Typ	Beschreibung
`id`	INTEGER PK	Eindeutige Dokument-ID (auto-increment)
`filename`	TEXT	Interner Dateiname (im docs/-Ordner)
`original_name`	TEXT	Originaler Dateiname (angezeigt in der Liste)
`file_path`	TEXT	Vollständiger Pfad zur Datei
`file_size`	INTEGER	Dateigrösse in Bytes
`file_hash`	TEXT	SHA-256 Hash (für Duplikaterkennung)
`mime_type`	TEXT	MIME-Typ (z.B. application/pdf, image/webp)
`page_count`	INTEGER	Anzahl Seiten (PDF/WebP)
`ocr_text`	TEXT	Extrahierter Volltext (OCR oder Plugin)
`category`	TEXT	Zugewiesene Kategorie
`tags`	TEXT	Tags als JSON-Array
`notes`	TEXT	Kurze Notiz zum Dokument
`note_data`	TEXT	Rich-Text-Notiz (JSON mit Text + Bildern)
`confidence`	REAL	Konfidenz der automatischen Kategorisierung (0.0–1.0)
`source`	TEXT	Herkunft: scan, import, link, note
`thumbnail`	TEXT	Pfad zum Vorschaubild
`starred`	BOOLEAN	Favorit (Ja/Nein)
`archived`	BOOLEAN	Archiviert (Ja/Nein)
`processed`	BOOLEAN	OCR/Indexierung abgeschlossen
`created_at`	TIMESTAMP	Erstellungszeitpunkt
`updated_at`	TIMESTAMP	Letzter Änderungszeitpunkt

Tabelle: search_index

Spalte	Typ	Beschreibung
`doc_id`	INTEGER FK	Verweis auf documents.id
`terms`	TEXT	Alle indexierten Terme (space-separiert, lowercase, normalisiert)

Tabelle: categories

Spalte	Typ	Beschreibung
`id`	INTEGER PK	Eindeutige Kategorie-ID
`name`	TEXT	Anzeigename (z.B. „Rechnung“)
`icon`	TEXT	Emoji-Icon (z.B. 📋)
`keywords`	TEXT	Schlüsselwörter für Auto-Kategorisierung (Komma-getrennt)

sort_order	INTEGER	Reihenfolge in der Sidebar
parent_id	INTEGER	Eltern-Kategorie (für Hierarchie, NULL = Hauptkategorie)