MDKV: A Multitrack Markdown Container for Structured, Portable Documents

Daniel Ari Friedman
ORCID: 0000-0001-6232-9096
daniel@activeinference.institute
August 10, 2025
DOI: 10.5281/zenodo.16790555

Abstract

Digital knowledge work increasingly demands documents that are simultaneously multilingual, multi-audience, and multi-channel. Traditional single-file Markdown struggles when the same canonical content must coexist with translations, commentary, references, code exemplars, and revision notes – each with distinct lifecycles and audiences. This paper introduces MDKV, a simple but rigorous multitrack Markdown container that packages a document's canonical content and auxiliary tracks into a single, portable `.mdkv` file. An `.mdkv` is a ZIP archive with a YAML `manifest.yaml` and a `tracks/` directory of UTF-8 Markdown files. MDKV provides a principled data model, validation, search, and export services, a CLI, and a small GUI for selective preview and live editing. We formalize the MDKV model, present its software architecture, explain round-trip export semantics, and evaluate design trade-offs against adjacent technologies. The MDKV initial specification is open source under the Apache-2.0 license and maintained at the project repository (`https://github.com/docxology/mdkv`). We also situate MDKV with respect to Matroska (MKV) as background on containerization: although the two are orthogonal in domain (text vs. multimedia), they share conceptual lineage in extensible container design. We conclude with use cases, implications for governance and reproducibility, and future work such as richer validation rules and alternative exporters. This paper is itself a Markdown file that renders into a valid MDKV. The built artifact is available at paper/paper.mdkv; combined exports are under paper/_bundle.

1. Introduction

Authoring, collaboration, and publishing workflows routinely require: (i) a single canonical text; (ii) multiple translations; (iii) audience-specific commentary and references; (iv) code listings synchronized with the text; and (v) review and revision notes. Maintaining such layers in separate files and folders creates cognitive load and increases the risk of divergence. The central question is how to preserve a document's conceptual unity while keeping its layers decoupled for editing, governance, and rendering.

MDKV addresses this need with a multitrack container for Markdown. Rather than overloading a single source with embedded annotations and language variants, MDKV separates concerns into tracks— `primary` (canonical), `translation`, `commentary`, `code`, `reference`, `media_ref`, and `revision` —and composes views at export time. The container format is intentionally modest: plain Markdown and YAML wrapped in ZIP, optimized for transparency, diff-friendliness, and longevity.

Design motivations and lineage. MDKV applies general container principles (clear manifest, independent tracks, robust round-trip) from adjacent domains to plaintext authoring. It favors human-inspectable, durable primitives over custom binary formats, enabling simple tooling and long-term readability. The initial public release standardizes a minimal spec and a reference implementation with CLI and GUI.

Contributions. (i) A precise model and container format with minimal but sufficient validation; (ii) a modular architecture separating model, storage, and services, surfaced via CLI and GUI; (iii) detailed use cases spanning multilingual publishing, layered collaboration, reproducible distribution, and governance; and (iv) guidance for cryptographic provenance and conformance.

2. Background: Containerization for Documents and Media

2.1 Matroska (MKV): History and Technical Context

Matroska (MKV) is an open multimedia container format announced in 2002 as a fork of the Multimedia Container Format project, notably adopting EBML to enable self-describing, extensible binary structures. Over two decades, MKV matured into a widely supported container capable of packaging multiple video, audio, and subtitle streams, chapters, and rich metadata. Notable milestones include early stable releases, the emergence of WebM as a web-oriented subset, and native support in mainstream operating systems, underscoring the robustness and adaptability of a stream-oriented, metadata-rich container design. See also the official overview at matroska.org for goals and feature summaries.

Technically, MKV exemplifies key container principles: a flexible root structure (via EBML), multiplexing of heterogeneous tracks, and resilience features that permit partial playback in the presence of corruption. These characteristics make MKV a useful conceptual antecedent for any domain where multiple orthogonal tracks must be co-packaged without imposing codec-level coupling.

Key timeline highlights:

- 2004: First stable Matroska releases signaled production maturity.
- 2010: Google introduced WebM as a web-delivery subset of Matroska, emphasizing open codecs and browser playback.
- 2014: Microsoft announced native MKV support in Windows 10, cementing broad OS-level adoption.

2.2 MDKV and MKV: Orthogonal Domains, Shared Intuition

MDKV is not an MKV editor, wrapper, or derivative; it targets plaintext documents rather than multimedia. The similarity in names is intentional only insofar as both are "container-centric" designs. MDKV borrows the intuition that many artifacts are multi-track by nature, but it embraces a radically simpler substrate—Markdown (CommonMark), YAML, and ZIP—to maximize openness, inspectability, and compatibility with standard tooling. Where MKV multiplexes time-based streams, MDKV co-packages logically distinct textual layers.

2.3 Why a textual container (vs. multimedia/EPUB/PDF)

- **Keep primitives simple**: plain Markdown + YAML + ZIP are ubiquitous, inspectable, and durable; no custom binary framing.
- **Multi-track authorship, not playback**: MDKV is for co-authoring and distributing textual layers in parallel; it is not a media streaming format.
- **Deterministic composition**: exports are reproducible from a manifest and track set, supporting provenance and verification.
- **Toolchain leverage**: downstream pipelines (e.g., pandoc) can consume `to_markdown()` outputs directly.
- **Manage external assets**: media stays outside via `media_ref` tracks, keeping bundles lean while links remain explicit and reviewable.
- **Human-first diffs**: separate files per track minimize merge conflicts and make reviews targeted.

- **Sovereign metadata**: `manifest.yaml` is explicit and auditable; no hidden state.

3. MDKV Concept and Format

3.1 Overview

An `.mdkv` file is a ZIP archive that contains:

- `manifest.yaml` : top-level document metadata and an index of tracks
- `tracks/` : UTF-8 Markdown files, one per track

The format is purposely simple for inspection and tooling: no custom filesystem or binary framing beyond ZIP, and no Markdown dialect beyond CommonMark-compatible text.

3.2 Manifest Schema

Top-level fields (required unless noted):

- `title` : string
- `authors` : list[string] (at least one)
- `created` : ISO-8601 datetime string
- `version` : string (optional; default "0.1")
- `metadata` : object<string,string> (optional)
- `tracks` : list[Track] (optional at the schema level, but at least one `primary` track is required by validation)

Track object fields:

- `track_id` : string (unique within the document)
- `track_type` : one of `primary`, `translation`, `commentary`, `code`, `reference`, `media_ref`, `revision`
- `language` : string or null (BCP-47/ISO-639 recommended for linguistic content)
- `path` : string (must start with `tracks/` ; paths SHOULD end with `.md`)

Example manifest:

```
 title: Example
authors: ["Author"]
  created: 2025-01-01T00:00:00Z
version: "0.1"
metadata: {}
tracks:
  - track_id: primary
    track_type: primary
    language: en
    path: tracks/primary.md
  - track_id: notes
    track_type: commentary
    language: null
    path: tracks/notes.md
```

3.2.1 Specification at a glance

Field overview:

Field	Type	Required	Notes
`title`	string	yes	Document title
`authors`	list[string]	yes	At least one author
`created`	ISO-8601 string	yes	UTC recommended
`version`	string	no	Default "0.1"
`metadata`	map[string,string]	no	Freeform key/value pairs
`tracks`	list[Track]	schema-optional	Validation requires at least one `primary`

Track descriptor overview:

Field	Type	Required	Notes
`track_id`	string	yes	Unique within the document
`track_type`	enum	yes	One of `primary, translation, commentary, code, reference, media_ref, revision`
`language`	string or null	no	BCP-47 recommended for linguistic tracks
`path`	string	yes	Must start with `tracks/`

3.3 Multi-track example (manifest + track bodies)

Below is a compact, realistic multi-track container using several track types side-by-side. This illustrates how translations, commentary, code listings, references, media references, and revisions live next to the canonical `primary` track.

Manifest:

```
 title: Product Onboarding Guide
authors: ["Tech Docs Team"]
created: 2025-06-01T12:00:00Z
version: "0.1"
metadata:
  audience: "customer, internal"
tracks:
  - track_id: primary
    track_type: primary
    language: en
    path: tracks/primary.md
  - track_id: ru
    track_type: translation
    language: ru
    path: tracks/ru.md
  - track_id: zh
    track_type: translation
    language: zh
    path: tracks/zh.md
  - track_id: commentary
    track_type: commentary
    language: null
    path: tracks/commentary.md
  - track_id: snippets
    track_type: code
    language: null
    path: tracks/snippets.md
  - track_id: refs
    track_type: reference
    language: null
    path: tracks/refs.md
  - track_id: media
    track_type: media_ref
    language: null
    path: tracks/media.md
  - track_id: release_notes
    track_type: revision
    language: null
    path: tracks/release_notes.md
```

Track bodies (excerpts):

```
<!-- track:primary type:primary lang:en -->
# Welcome
Follow these steps to get started.

<!-- track:ru type:translation lang:ru -->
# Добро пожаловать
Выполните следующие шаги, чтобы начать.

<!-- track:zh type:translation lang:zh -->
# 欢迎
请按照以下步骤开始。

<!-- track:commentary type:commentary lang:None -->
Editors: confirm screenshots match v1.3 UI before release.

<!-- track:snippets type:code lang:None -->
```bash
uv run mdkv init --title "Guide" --author "Docs" --out guide.mdkv
```

<!-- track:refs type:reference lang:None -->
- Quickstart video: media:gui_demo.webm
- Markdown spec: https://commonmark.org

<!-- track:media type:media_ref lang:None -->
assets/gui_demo.webm # external file shipped alongside exported package

<!-- track:release_notes type:revision lang:None -->
- 2025-06-15: Updated step 2 for new sign-in flow.
```

Combined exports simply concatenate these bodies in the order present, preserving the headers for round-trip reconstruction. Composition order follows the manifest track list order.

3.4 This paper as an MDKV (self-contained guidance)

This Markdown file is authored to be round-trip friendly for MDKV tooling. To package it as a `.mdkv`:

1. Create `manifest.yaml` with at least one primary track and metadata:

```
title: MDKV: A Multitrack Markdown Container for Structured, Portable Documents
authors: ["Daniel Ari Friedman"]
created: 2025-08-10T00:00:00Z
version: "0.1"
metadata:
  doi: "10.5281/zenodo.16790555"
  orcid: "0000-0001-6232-9096"
  contact: "daniel@activeinference.institute"
tracks:
  - track_id: primary
    track_type: primary
    language: en
    path: tracks/primary.md
```

1. Place this file's Markdown content (including the `<!-- track:... -->` header above) into `tracks/primary.md`.
2. Zip `manifest.yaml` and `tracks/` into `paper.mdkv` (use the CLI below), then validate.

CLI (linked): see `docs/cli.md`. Minimal sequence:

```
uv run mdkv init --title "MDKV Paper" --author "Daniel Ari Friedman" --out paper.mdkv
uv run mdkv update-track paper.mdkv --id primary --content "$(cat paper/mdkv_paper.md)"
uv run mdkv set-meta paper.mdkv doi 10.5281/zenodo.16790555
uv run mdkv validate paper.mdkv
uv run mdkv info paper.mdkv
```

3.5 Validation Rules

MDKV's base validator enforces:

- Presence of `title`, `authors`, and at least one `primary` track
- Track `path` begins with `tracks/`
- `track_type` is one of the supported values
- `track_id` is non-empty and unique
- `language` is either a BCP-47 string or null

The intentionally small rule set preserves extensibility while ensuring minimal integrity guarantees for downstream services.

Implementation note: the reference validator currently checks for the presence of a track with id `primary`; this could be aligned to the type-level requirement in future releases.

3.6 Round-Trip Export Headers

When exporting multiple tracks into a single Markdown stream, MDKV prefixes each track with a lightweight HTML comment that encodes track metadata. These hints enable round-trip attribution and reconstruction without altering the track files in the container:

```
<!-- track:primary type:primary lang:en -->
Title

<!-- track:notes type:commentary lang:None -->
Editorial notes here
```

3.7 Normative Container Requirements (Conformance)

The MDKV container is defined by the following MUST/SHOULD requirements:

- A valid `.mdkv` MUST be a ZIP archive containing a root-level `manifest.yaml` and a `tracks/` directory.
- `manifest.yaml` MUST be a valid YAML mapping with fields: `title` (string), `authors` (list of strings, non-empty), `created` (ISO-8601 string), and optionally `version` (string, default "0.1"), `metadata` (mapping), and `tracks` (list of track descriptors).
- Each track descriptor MUST include `track_id` (unique string), `track_type` (one of the allowed values), `language` (string or null), and `path` (string beginning with `tracks/`).
- At least one track with `track_type: primary` MUST be present.
- For round-trip combined Markdown exports, each included track MUST be prefixed by an HTML comment header of the form: `<!-- track:{id} type:{type} lang:{lang} -->`.
- Implementations SHOULD preserve field ordering in `manifest.yaml` for human readability, though ordering is not semantically significant.
- Implementations SHOULD emit UTF-8 for all track files and SHOULD NOT assume line ending conventions beyond standard text processing.

3.8 Conformance Levels

- Core Conformance: satisfies all MUST requirements above and passes validation.
- Extended Conformance: additionally supports selective export by track type and language filters, and preserves track headers for round-trip workflows.

3.9 Design principles

- Simplicity first: rely on ubiquitous primitives (ZIP, YAML, Markdown) to maximize longevity and inspectability.
- Separation of concerns: keep tracks orthogonal; orchestrate at export time rather than embedding format-specific directives into bodies.
- Round-trip fidelity: preserve attribution via headers so combined streams can be losslessly de-multiplexed.
- Deterministic composition: identical manifests and track sets produce identical exports to support reproducible publishing.
- Extensibility: add tracks and exporters without changing the core model or on-disk layout.

3.10 Limitations of the format

- **No encryption or signing baked in**: use external tools for security (see Security and provenance guidance).
- **Not an execution environment**: `code` tracks are listings, not runnable notebooks; execution belongs to downstream systems.
- **No enforced cross-track alignment**: paragraph-level alignment between tracks is a convention, not a rule.
- **Binary assets live outside**: large media is referenced via `media_ref` instead of embedded to keep bundles lean and editable.
- **No cross-doc linking semantics**: references across containers are conventional; not part of the base spec.
- **Time semantics are shallow**: `created` is recorded; versioning/history are external concerns.

4. Software Architecture

MDKV follows a thin-orchestrator, modular architecture:

- `mdkv.core`: data model and validation
- `mdkv.storage`: persistence (read/write `.mdkv`)
- `mdkv.services`: search and export utilities
- `mdkv.cli`: CLI entry points and subcommands

This separation enables stable public APIs, testable modules, and straightforward extension.

4.1 Core Model (`mdkv.core`)

The core defines two primary classes:

- `Track`: a single Markdown track with `track_id`, `track_type`, `language`, `path`, `content`, and invariant checks (valid type, non-empty id, `path` under `tracks/`).
- `MDKVDocument`: an in-memory aggregate with metadata (`title`, `authors`, `created`, `version`, and `metadata` map) and a mapping `track_id → Track`. It provides helpers for adding/removing tracks, renaming, content updates, language listing, and metadata set/get/remove.
- Invariants are validated eagerly on construction and update; violations raise narrow exceptions (`ValueError`, `ValidationError`).

The core also surfaces `allowed_track_types()` and validation types (`ValidationError`, `ValidationIssue`).

4.2 Storage (`mdkv.storage`)

ZIP + YAML persistence:

- `save_mdkv(doc, output_path)`: write tracks under `tracks/` and emit `manifest.yaml` from the document.
- `load_mdkv(input_path)`: parse the manifest and reconstruct the document, loading each track's content.

The manifest lists metadata and track descriptors; bodies remain discrete files for transparency and editability.

4.3 Services (`mdkv.services`)

Two primary services operate over `MDKVDocument` instances:

- Search: regex search with optional filters by `track_type` and `language`, returning match structures suitable for CLI/GUI display.
- Export: composition of multi-track Markdown via `to_markdown()` and primary-track HTML via `to_html()` using markdown-it-py. `export_to_files()` writes selected tracks to individual `.md` files for channel-specific distribution.
- Export determinism: order is manifest-driven; identical inputs produce identical outputs.

4.4 CLI (`mdkv.cli`)

Thin orchestrators for: init, info, validate, track ops, search, export, metadata, and GUI. Commands call module APIs directly for testability and minimal command-layer logic.

4.5 GUI (`mdkv.gui`)

The optional GUI offers a live preview and editing workflow with track selection via checkboxes. A backend endpoint (`POST /api/render/tracks_html`) renders arbitrary track subsets, while the visual editor maintains a full combined view with live persistence per track. Additional endpoints include `GET /api/render/html` (primary HTML) and `GET /api/render/markdown` (combined Markdown). The GUI is intended as an affordance for users uncomfortable with the CLI while preserving parity with programmatic workflows. State is ephemeral; persistence uses the same `save_mdkv` path as the CLI.

4.6 Architecture overview (linked)

- Layers and responsibilities: see `docs/architecture.md` and the published docs site.
- Core model: `mdkv/core/model.py` defines `Track`, `MDKVDocument`, and invariants; re-exported via `mdkv/__init__.py`.
- Storage: `mdkv/storage/io.py` implements `save_mdkv` and `load_mdkv` (ZIP + YAML manifest and `tracks/`).
- Services: export in `mdkv/services/export.py` (`to_markdown`, `to_html`, `export_to_files`); search in `mdkv/services/search.py`. Both are re-exported at the top-level package for a stable public API.
- CLI: `mdkv/cli/main.py` surfaces subcommands (init, info, validate, track ops, search, export, gui).
- GUI backend: `mdkv/gui/server.py` provides `POST /api/render/tracks_html` for subset rendering; static assets under `mdkv/gui/static/`.

Data flow summary (see also the repository `README.md` and docs pages):

- Authoring: create or load an `MDKVDocument`, add tracks; validate.
- Persistence: `save_mdkv` writes `tracks/` and `manifest.yaml` into a `.mdkv` ZIP; `load_mdkv` reconstructs.
- Services: `search_document` filters/matches; `to_markdown` and `to_html` export views; `export_to_files` writes per-track files. For end-to-end CLI usage, see `docs/cli.md`.

5. Public API and Re-Exports

The top-level `mdkv` package re-exports the public API to provide a stable surface:

- `MDKVDocument`, `Track`, `allowed_track_types`, `ValidationError`, `validate_document`, `ValidationIssue`
- `search_document`, `SearchMatch`, `to_markdown`, `to_html`, `export_to_files`
- `save_mdkv`, `load_mdkv`

This consolidation supports both script-level usage and library integrations without deep import paths.

6. Usage

6.1 Quickstart

```
# install uv once per machine (Linux/macOS)
curl -LsSf https://astral.sh/uv/install.sh | sh

# from repo root
uv venv
uv run pytest -q
uv run mdkv --help
```

6.2 Create, Inspect, Validate

```
uv run mdkv init --title "Doc" --author "You" --out doc.mdkv
uv run mdkv info doc.mdkv
uv run mdkv validate doc.mdkv
```

6.3 Track Operations

```
uv run mdkv list-tracks doc.mdkv
uv run mdkv add-track doc.mdkv --id notes --type commentary --lang "" --content "Note"
uv run mdkv rename-track doc.mdkv --old-id notes --new-id commentary
uv run mdkv update-track doc.mdkv --id commentary --content "Updated note"
```

6.4 Export and GUI

```
# export selected track types to Markdown
uv run mdkv export-tracks doc.mdkv --types primary,commentary > exported.md

# export HTML of primary track
uv run mdkv export --html doc.mdkv > primary.html

# launch GUI
uv run mdkv gui --path doc.mdkv
```

6.5 Programmatic Export

```
from pathlib import Path
from datetime import datetime
from mdkv import MDKVDocument, Track, export_to_files, to_markdown, save_mdkv, validate_document

# write selected tracks as individual .md files
doc = MDKVDocument(title="Doc", authors=["You"], created=datetime.utcnow(), version="0.1")
doc.add_track(Track(track_id="primary", track_type="primary", language="en", path="tracks/primary.md", content="# Hello\n"))

# validate, save, export
validate_document(doc)
save_mdkv(doc, Path("doc.mdkv"))
export_to_files(doc, Path("out_tracks"), include_track_types=["primary", "commentary"])

# combined markdown (for pandoc or review)
combined_md = to_markdown(doc)
Path("combined.md").write_text(combined_md, encoding="utf-8")
```

6.6 End-to-end: create a multi-track paper

```
# 1) Initialize
uv run mdkv init --title "Paper" --author "Author" --out paper.mdkv

# 2) Add tracks
uv run mdkv add-track paper.mdkv --id ru --type translation --lang ru --content "# Резюме\n..."
uv run mdkv add-track paper.mdkv --id zh --type translation --lang zh --content "# 摘要\n..."
uv run mdkv add-track paper.mdkv --id commentary --type commentary --content "Editors: tighten abstract."
uv run mdkv add-track paper.mdkv --id code --type code --content "```bash\nuv run pytest -q\n```"
uv run mdkv add-track paper.mdkv --id refs --type reference --content "- CommonMark: https://commonmark.org"
uv run mdkv add-track paper.mdkv --id media --type media_ref --content "docs/_static/gui_demo.webm"
uv run mdkv add-track paper.mdkv --id revisions --type revision --content "- 2025-08-10: camera-ready edits"

# 3) Inspect and validate
uv run mdkv list-tracks paper.mdkv | jq .
uv run mdkv validate paper.mdkv

# 4) Search (only Russian translations for TODO/FIXME)
uv run mdkv search paper.mdkv --pattern "TODO|FIXME" --types translation --languages ru | jq .

# 5) Export views
uv run mdkv export-tracks paper.mdkv --types primary,translation > combined.md
uv run mdkv export --html paper.mdkv > primary.html
```

7. Data Model: Semantics of Tracks

MDKV's track taxonomy encodes common document layers:

- `primary` : canonical content intended for publication and primary consumption.
- `translation` : language alternatives to `primary` , preserving structure but adapting language and, where necessary, locale.
- `commentary` : annotations, editorial notes, and audience-specific guidance; excluded by default from canonical exports.
- `code` : code listings and examples that remain synchronized with the narrative; not intended as executed notebooks.
- `reference` : citations and reference lists, metadata-heavy but typically orthogonal to `primary` structure.
- `media_ref` : references to external media (figures, videos) that are managed outside the container but linked textually.
- `revision` : review notes and change summaries; useful for governance, release notes, and compliance trails.

Semantically, MDKV treats tracks as parallel layers; it does not prescribe alignment at paragraph or section granularity. Alignment strategies (e.g., anchor conventions) can be layered atop the format by convention when needed.

8. Validation and Integrity

Minimal by design to maximize compatibility while catching common integrity issues. Optional profiles can enforce stricter policies (e.g., translation coverage, mandatory references) without compromising baseline portability. See Appendix C for an example of bundle metadata and checksums emitted by the paper builder.

8.1 Common validation errors and remedies

- **Missing `primary` track**: Add at least one `primary` track; ensure `track_type: primary` .
- **Invalid track `path`** : Ensure `path` starts with `tracks/` and corresponds to a file in the container.
- **Unknown `track_type`** : Use only the allowed set; see section 7.
- **Empty `authors`** : Provide at least one author string.
- **Malformed `created`** : Use ISO-8601 (e.g., `2025-08-10T00:00:00Z`).
- **Duplicate `track_id`** : Ensure each `track_id` is unique; rename or remove conflicts.
- **Bad `track_type` change**: When changing a track's type, ensure downstream exporters are aware of filtering.

9. Search and Export

Regex search across selected tracks and languages enables cross-cutting queries (e.g., beta-flagged commentary in Spanish). Export is deterministic given a manifest and track set:

- Multi-track Markdown: concatenates track bodies with track headers for round-trip attribution and optional filtering by track type.
- HTML (primary): renders the `primary` track to HTML for simple distribution channels, leveraging a standard Markdown engine.
- File export: writes track contents to discrete `.md` files for downstream pipelines.

These capabilities support reproducible publishing.

Programmatic search is available via `mdkv.services.search.search_document(doc, pattern, track_types=None, languages=None)` ; the CLI mirrors this with `uv run mdkv search <path> --pattern PATTERN [--types ...] [--languages ...]` . For combined exports intended for downstream tools (e.g., pandoc), keep round-trip headers intact to preserve provenance.

10. Implementation Notes

- Python 3.9+; dataclasses and type hints; explicit naming.
- Packaging: public API re-exports; CLI entry point `mdkv`.
- Dependencies: YAML, ZIP (std lib), Markdown renderer for HTML export.
- Tests: cover CLI, services, model invariants, and storage round-trip.

11. Use Cases

11.1 Multilingual publishing and localization

- Goal: ship a single canonical text to multiple locales without forking.
- Tracks: `primary` (en), `translation` (ru, zh, …).
- Flow: author in `primary`; translators add language tracks; QA searches for unresolved tags (e.g., TODO) scoped to translations; export per-locale Markdown/HTML.
- Benefit: eliminates redundant branching; preserves structure and provenance across locales.
- Notes: reserve `track_id` values for ISO language tags where possible (e.g., `ru`, `zh-Hans`).

11.2 Layered collaboration, review, and stakeholder tailoring

- Goal: separate canonical narrative from internal commentary and change rationale.
- Tracks: `commentary`, `revision` alongside `primary`.
- Flow: reviewers annotate in `commentary`; change summaries accrue in `revision`; public exports omit commentary; internal exports include it for audit and coaching.
- Benefit: clean public artifacts with rich internal discourse and governance trails.
- Notes: treat `revision` as append-only; include date stamps for traceability.

11.3 Developer documentation with synchronized code listings

- Goal: keep examples synced with docs and ship channel-specific bundles.
- Tracks: `code` for listings; optional `media_ref` for demos.
- Flow: update code listings in `code`; search across `code` and `primary` for API drift; export code-only bundles for IDE extensions and full docs for websites.
- Benefit: reduces drift between narrative and examples; supports tailored distribution.
- Notes: pair `code` listings with tested snippets in CI; consider tangling to source when appropriate.

11.4 Scholarly and technical writing with curated references

- Goal: bundle narrative with references and domain annotations.
- Tracks: `reference` for citations/bibliography; optional `commentary` for reviewer notes.
- Flow: maintain references in `reference`; export tight submissions without commentary; archive complete `.mdkv` with all layers for reproducibility.
- Benefit: transparent, reproducible artifacts while enabling audience-specific exports.
- Notes: adopt a citation style within `reference` tracks and keep DOIs/URLs explicit.

11.5 Governance, compliance, and audit trails

- Goal: capture intent and decision history without imposing a VCS workflow on recipients.
- Tracks: `revision` for change logs; `metadata` for release info.
- Flow: record rationale per release in `revision`; embed DOI, version, or policy IDs in `metadata`; export for auditors or regulators with selected layers.
- Benefit: lightweight provenance and traceability without heavy infrastructure.
- Notes: use checksums/signatures described in Appendix C to support integrity attestations.

11.6 Product, policy, and dual-audience documentation

- Goal: publish the same canonical content to internal and external audiences.
- Tracks: `commentary` (internal notes), `reference` (appendices), `media_ref` (assets).
- Flow: export public bundles from `primary` only; export internal bundles with commentary and references for onboarding/support teams.
- Benefit: single source of truth with audience-appropriate views.
- Notes: mark internal commentary with a prefix (e.g., `INTERNAL:`) to aid filtering.

11.7 Digital archiving and long-term preservation

- Goal: store durable, inspectable artifacts that migrate across tools.
- Tracks: all, with emphasis on `manifest.yaml` readability.
- Flow: write `.mdkv` for archives; include checksums and optional signatures; consumers can inspect and transform with standard tooling.
- Benefit: longevity and portability via ubiquitous primitives (ZIP, YAML, Markdown).
- Notes: include `BUNDLE_INFO.yaml` with source commit metadata when available.

12. Implications

- Governance: `revision` + metadata expose intent and review structure.
- Reproducibility: deterministic export + manifest versioning.
- Interoperability: ubiquitous primitives (ZIP, YAML, Markdown).
- Diff-friendliness: semantic separation reduces merge conflicts.
- Extensibility: add track types/exporters without core changes.
- Supply-chain clarity: explicit manifests and deterministic builds simplify reproducibility and review.

Licensing and community

- **License**: Apache-2.0. See LICENSE in the repository.
- **Source & issues**: development and issue tracking on the GitHub repository.
- **Contributions**: small, focused pull requests preferred; tests and docs alongside changes.

Security and Privacy Considerations

- `.mdkv` bundles are not encrypted; sensitive content SHOULD be managed with appropriate transport and storage controls (e.g., encryption at rest, signed artifacts).
- `metadata` may contain personally identifiable information; producers SHOULD minimize PII and consumers SHOULD handle metadata under applicable policies.
- Track headers in exported Markdown are informational; avoid embedding secrets in headers or comments.
- Validation is minimal by design; distributors SHOULD layer additional checks where provenance and integrity are critical (e.g., checksums, signatures).

Cryptographic provenance and signing

- Manifest and track checksums: producers MAY compute SHA-256 (or stronger) digests for each file and include them in `metadata` for integrity verification after transport.
- Container signing: producers MAY sign the `.mdkv` archive (e.g., using minisign or GPG) and distribute the signature alongside the artifact. Consumers SHOULD verify signatures prior to ingest in security-sensitive contexts.
- Reproducible builds: deterministic export semantics enable consistent digests across builds from the same manifest/tracks. Consumers SHOULD compare published checksums against local builds when a source repository is available.
- Threat model notes: MDKV is plaintext-oriented and does not execute code; however, downstream pipelines (e.g., HTML renderers) SHOULD sanitize untrusted Markdown before display.

Interoperability and Migration Guidance

- Content SHOULD be authored in CommonMark-compatible Markdown to maximize renderer compatibility.
- For downstream channels (PDF/EPUB), use `to_markdown()` with pandoc-compatible flags/templates; retain track headers to assist provenance.
- When migrating legacy multi-file projects, prefer one-track-per-file mapping under `tracks/` and encode provenance in `metadata`.

13. Related Work and Positioning

- Matroska (MKV): multimedia container with EBML, multiplexing heterogeneous streams and metadata. Conceptually adjacent as a container design; orthogonal in domain (binary streams vs. plaintext tracks).
- CommonMark: Markdown specification underpinning content rendering; MDKV does not introduce dialect changes.
- EPUB/PDF pipelines: rich, distribution-focused bundles; MDKV targets authoring/editing unity and channel-agnostic composition, and can serve as an upstream source for such exporters.
- Jupyter notebooks: literate, executable documents combining code and text; MDKV focuses on non-executable, layered textual artifacts with optional code listings.

14. Future Work

- Validation profiles: optional, stricter validators (coverage constraints, reference bibliography schemas).
- Alternate exporters: PDF/EPUB builders atop `to_markdown()`, pandoc interop, and track-aware templates.
- Track alignment aids: soft anchors and cross-track link conventions for advanced localization/review.
- Provenance and signing: cryptographic checksums and signatures for manifests and tracks.
- Remote storage: mapping tracks to object stores while preserving a virtual manifest.
- Richer GUI: per-track editing affordances, diff views, and translation workflows.

15. Conclusion

MDKV introduces a practical, minimal, and extensible approach to multitrack textual documents. By embracing simple, durable primitives (Markdown, YAML, ZIP) and separating core modeling from storage and services, MDKV enables multilingual and multi-audience authoring without branching, improves governance through explicit layer semantics, and supports reproducible publishing. As the ecosystem evolves, MDKV can integrate stronger validation profiles and richer exporters while preserving its core promise: a small, inspectable container for structured documents that travels well and ages gracefully.

References

- Matroska (MKV) overview: Wikipedia
- CommonMark specification: commonmark.org
- YAML 1.2 specification: yaml.org/spec
- ZIP Appnote (PKWARE): PKWARE Appnote
- markdown-it-py project: executablebooks/markdown-it-py
- Astral uv project: astral-sh/uv

Appendix A: Glossary

- MDKV: Multitrack Markdown container defined by this specification.
- Track: A named, typed Markdown layer within an MDKV document (e.g., `primary`, `translation`).
- Manifest: YAML file at the root of the container that indexes tracks and captures metadata.
- Round-trip: Ability to reconstruct track attribution from a combined export using embedded headers.
- Conformance: Degree to which an implementation satisfies normative requirements in this document.

Appendix B: Track header grammar

The header used for round-trip exports follows a simple, stable grammar:

```
<!--\s+track:{track_id}\s+type:{track_type}\s+lang:{language}\s+-->
```

When language is unspecified, the literal text `None` is emitted for `{language}`.

6.7 Pandoc pipeline (PDF)

```
# render primary track to Markdown, then to PDF with pandoc
uv run mdkv export paper.mdkv > primary.md
pandoc primary.md -o primary.pdf
```

6.8 Inspect and verify

```
# show manifest summary and tracks
uv run mdkv info paper.mdkv | sed -n '1,60p'

# validate and list tracks
uv run mdkv validate paper.mdkv
uv run mdkv list-tracks paper.mdkv
```

6.9 When to choose MDKV

- **You need layered authorship**: separate canonical text, translations, commentary, references, code listings, and revisions without branching the project.
- **You want reproducible bundles**: deterministic exports and manifest-driven structure make builds auditable.
- **You prefer plain text pipelines**: leverage Markdown/YAML and existing tools (git, grep, pandoc) rather than custom ecosystems.
- **You need selective exports**: tailor output by track types and languages for different audiences/channels.
- **You need to be able to edit the document via multiple tools**: MDKV is a container for text, not a text editor.
- **You want to use an open source format to ensure long-term compatibility**: MDKV is a small, simple, modular, and well-documented format that is licensed under the Apache 2.0 license.
- `track_id`: string, unique per document
- `track_type`: one of `primary, translation, commentary, code, reference, media_ref, revision`
- `language`: BCP-47 code or `None`

Appendix C: Example bundle artifacts

This appendix sketches the files created by the builder (`paper/build_paper_bundle.py`) to support reproducible publication and verification.

- `paper/paper.mdkv`: ZIP archive with `manifest.yaml` and `tracks/` (e.g., `tracks/primary.md`, optional `tracks/references.md`).
- `paper/_bundle/paper.md`: combined multi-track Markdown export from `to_markdown()`.
- `paper/_bundle/paper.html`: HTML of the `primary` track from `to_html()`.
- `paper/_bundle/tracks/`: individual per-track Markdown files exported for distribution pipelines.
- `paper/_bundle/BUNDLE_INFO.yaml`: bundle metadata (source path, timestamps, clone info, commit hashes when available).
- `paper/_bundle/CHECKSUMS.yaml`: SHA-256 digests of `paper.mdkv` and exported files to enable integrity checks.

Example excerpt of `BUNDLE_INFO.yaml`:

```
generated_at: 2025-08-10T12:34:56Z
source_markdown: /abs/path/paper/mdkv_paper.md
output_mdkv: /abs/path/paper/paper.mdkv
repo_url: https://github.com/docxology/mdkv
cloned_commit: 0123abcd...
builder_commit: 89ef4567...
```

Example excerpt of `CHECKSUMS.yaml`:

```
_bundle/paper.html: 7f4c2b...
_bundle/paper.md: 8b91fe...
_bundle/tracks/primary.md: 0b12aa...
paper.mdkv: 5c44d1...
```