

≡ ⌂ > ChatGPT 5.2 ▾

🕒 ⏚ ⏷ ⋮

Given what we have discussed about truthdb, how would you

Your question is incomplete. Please finish it.

🔗 🤗 🤔 ⏪ ⋮

write a requirement document about the WAL.

Below is a **requirements document for the TruthDB Write-Ahead Log (WAL)**.

It is written as a **design-level requirements spec**, not an implementation doc.

TruthDB – Write-Ahead Log (WAL) Requirements

1. Purpose & Scope

The TruthDB WAL is the **authoritative durability and ordering mechanism** for all state changes in TruthDB.

It guarantees **crash safety**, **deterministic replay**, and **high-throughput sequential writes** while serving as the foundation for:

- Event-sourced subsystems
- Replication and recovery
- Deterministic rebuild of derived state
- Time-ordered auditing and inspection

The WAL is **not** a query engine, index, or storage format for materialized state. It is a **log**.

2. Core Principles

1. Append-only
2. Sequential I/O optimized
3. Crash-consistent
4. Deterministic replay

5. Minimal abstraction leakage

6. Explicit over implicit

3. Functional Requirements

3.1 Append Semantics

- WAL entries MUST be appended sequentially.
 - No in-place modification is allowed.
 - No compaction, deletion, or rewriting occurs inside active WAL segments.
-

3.2 Durability Guarantees

- WAL MUST support configurable durability modes:
 - `async` (OS buffered)
 - `sync` (`fsync` on commit boundary)
 - `group_sync` (batched `fsync`)
- A committed entry MUST survive:
 - process crash
 - kernel panic
 - power loss (assuming storage honors `fsync` semantics)

3.3 Ordering & Identity

- Every WAL entry MUST have:
 - a **monotonically increasing sequence number**
 - a **logical timestamp** (not wall-clock dependent)
 - Ordering MUST be total within a WAL stream.
-

3.4 Entry Structure

Each WAL entry MUST contain:

- Header:
 - `version`
 - `entry type`
 - `payload length`

- checksum
- Payload:
 - opaque bytes (TruthDB-level encoding)
- Footer (optional):
 - duplicate length or checksum (for torn-write detection)

WAL MUST NOT interpret payload contents.

3.5 Checksums & Corruption Detection

- Every entry MUST be protected by a fast checksum (e.g. xxHash64).
 - WAL replay MUST:
 - detect partial writes
 - stop safely at the last valid entry
 - Silent corruption MUST NOT propagate into state replay.
-

4. Segmentation & File Layout

4.1 WAL Segments

- WAL MUST be split into **fixed-size segments**.
 - Segment size MUST be configurable at build or runtime.
 - Segment rollover MUST be deterministic.
-

4.2 Segment Lifecycle

Each segment is in exactly one state:

1. **Active** (currently being written)
2. **Sealed** (complete, immutable)
3. **Archived** (eligible for retention / snapshot pruning)

No segment may transition backwards.

4.3 File Naming & Discovery

- Segment filenames MUST encode:
 - starting sequence number
 - segment index

- WAL MUST be discoverable via directory scan only.
 - No external metadata is required to locate the WAL.
-

5. Replay Requirements

5.1 Replay Determinism

- Given the same WAL bytes, replay MUST produce identical results.
 - Replay order MUST be strictly defined.
 - Replay MUST be independent of wall-clock time.
-

5.2 Partial Replay

- WAL MUST support replay from:
 - start
 - arbitrary sequence number
 - segment boundary
 - Replay MUST stop cleanly on corruption.
-

6. Concurrency Model

6.1 Writers

- WAL MUST support:
 - single-writer (baseline)
 - multi-producer → single-writer queue
 - No concurrent file writes are allowed.
-

6.2 Readers

- WAL MUST support concurrent readers:
 - tailing readers
 - snapshot readers
 - Readers MUST NOT block writers.
-

7. Performance Requirements

7.1 Write Path

- WAL MUST be optimized for:
 - large sequential writes
 - minimal syscalls
 - cache-friendly batching
 - No allocation on the hot path after initialization.
-

7.2 Backpressure

- WAL MUST expose backpressure signals:
 - queue depth
 - fsync latency
 - disk saturation
 - TruthDB MAY choose to reject writes rather than buffer unboundedly.
-

8. Integration Requirements

8.1 Snapshotting

- WAL MUST support coordination with snapshots:
 - snapshot at sequence N
 - WAL segments < N become reclaimable
 - WAL itself does not create snapshots.
-

8.2 Replication & Streaming

- WAL MUST be streamable:
 - byte-exact
 - order-preserving
 - Remote replication MUST NOT require WAL interpretation.
-

9. Failure Semantics

9.1 Crash Recovery

On startup, WAL recovery MUST:

1. Scan segments in order

2. Validate entries
3. Stop at first invalid entry
4. Expose last committed sequence number

No repair attempts are allowed silently.

9.2 Corruption Policy

- Corruption MUST be:
 - detected
 - reported
 - isolated
 - WAL MUST NOT attempt heuristic recovery without explicit operator action.
-

10. Non-Goals

The WAL explicitly does **not**:

- Enforce schemas
 - Perform deduplication
 - Compact itself
 - Manage retention policy
 - Interpret events
 - Guarantee cross-stream ordering
-

11. Invariants (Must Always Hold)

- WAL is append-only
 - Entries are immutable once committed
 - Replay is deterministic
 - Corruption is detectable
 - Ordering is explicit and total
-

If you want next steps, we can:

- derive the **exact binary format**
- define **commit / transaction boundaries**
- split WAL into **logical streams**

- map this onto **io_uring + direct I/O**
- or compare this against **TigerBeetle / Kafka WAL designs**

