Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0004
Format Contract

Zaxonlite Format and Compatibility Contract

STATE
committed

CREATED
2026-07-20

LAST UPDATED
2026-07-21

INTENDED STATUS
Normative

AUTHORS
paxos-zig project

DISCUSSION

Frozen first-release durable and wire formats, normative for upgrade checks

LABELS
architecture, zaxonlite, formats

Status of This Memo

This document is an internal Zaxon discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

Contents

1. Status and scope

This document freezes the formats used by the first Zaxonlite release. It is normative for upgrade checks. Consensus safety does not permit a decoder to guess, skip an unknown durable record, or silently negotiate down to an unauthenticated protocol.

Supported production matrix: POSIX systems whose file and parent-directory `fsync` semantics meet the storage contract, and Windows 10 release 1809 or Server 2019 and newer on NTFS, under the durability model in ZDS 0006. A node probes the volume at startup and refuses to open where those semantics are missing. Unix socket listeners remain POSIX-only.

2. Compatibility policy

1. Wire compatibility is exact-major: protocol version 6 accepts only version 6. A rolling upgrade that changes the wire version requires an explicitly dual-version bridge release; there is no automatic downgrade.
2. Durable formats are versioned independently. Readers must reject a newer identity, journal, payload, descriptor, or snapshot format before voting.
3. Additive JSON response fields are compatible. Clients must ignore fields they do not use. Removed fields and changed meanings require a public API version change.
4. The Zig source API is release-candidate quality. The C ABI uses opaque handles and fixed-width types; incompatible ABI changes require a new symbol suffix or major library version.
5. Back up every node and verify at least one logical backup before an upgrade. A downgrade is supported only when the older binary declares every installed format and wire version compatible.

3. Network protocol v6

Every plain frame begins with little-endian `u32 total_after_length`, followed by `u8 kind` and a body. The bound is 64 MiB and zero-length frames are invalid. The first frame is an unprotected `hello` containing version, connection kind, node ID, database ID, and configuration ID.

The production profile carries these frames inside a TLS 1.3 mutual-authentication channel. An optional PSK provider adds a responder challenge with a fresh 32-byte nonce and `HMAC-SHA256(secret, domain || hello || nonce)`. The initiator verifies it and returns the corresponding client-domain proof. Both derive a connection key using a third domain. Every later body is:

```
u64 sequence_le || application_body || hmac_sha256
```

The MAC covers a frame domain, frame kind, sequence, and body. Each direction starts at sequence zero and accepts exactly the next sequence. A mismatch, invalid tag, missing proof, or older peer closes the connection. The optional PSK layer itself supplies authentication and integrity, not encryption. The production TLS channel supplies confidentiality and per-node identity. The explicit development profile may omit TLS only with a PSK provider and only when the listener and every peer use numeric loopback; leader redirects in that profile remain confined to caller-supplied seeds.

Protocol v6 retains the receiver-side payload-storage invariant. A sender queues `payload_data` immediately before a dependent envelope on the ordered stream; the receiver cannot enter Paxos until the object is verified and stored, and independently gates reordered/missing-payload envelopes. Backup streaming uses begin, ordered chunk, and end frames with an end-to-end SHA-256. Version 4 added `learner_commit`: a configured voter certifies one chosen slot to a non-

voting learner after that learner has durably acknowledged the payload. `learner_heartbeat` carries the leader's decided slot for bounded-staleness checks and is never treated as a promise, accepted vote, or commit. Version 5 adds the retained checkpoint proof and voter-digest probe/reply used before a cross-epoch snapshot install. Version 6 adds the bounded enrollment connection kind and its one-request token/CSR exchange.

3.1. Enrollment records and exchange

Enrollment is a narrow bootstrap for a node already present in the static registry. It does not add a member or grant a database role. An issuer is a normal mTLS node explicitly started with an owner-only CA private-key file. Creating a token remains an ordinary authenticated client RPC, so an unauthenticated caller cannot mint credentials.

The opaque `ZXET` version 1 bundle contains the target node ID, issuer node ID, database ID, expiry time, a random 256-bit secret, the issuer endpoint, and the complete cluster CA certificate. The file is owner-only and is a bearer secret. Its maximum encoded size is 128 KiB; the endpoint is at most 512 bytes and the CA PEM at most 64 KiB. The issuer persists only a domain-separated SHA-256 hash of the secret in an owner-only `ZXER` version 1 record under `enrollment-tokens/`, together with the three identity bindings and expiry.

The joiner pins the bundled CA and the exact issuer common name, generates a P-256 private key and signed CSR locally for `zaxon-node-<target-id>`, and sends one `enrollment_request`: 32-byte secret, target node ID, database ID, CSR length, and at most 16 KiB of CSR PEM. This is the sole production TLS path on which the server may accept a connection without a client certificate. The application frame must be an enrollment hello followed by exactly that one request; all other connection kinds still require a verified client certificate. The response is a one-byte status and, on success, at most 64 KiB of certificate PEM.

Before signing, the issuer verifies the request/hello identity, database, static registry, revocation state, CSR signature, and exact common name. It then atomically renames the token record from `.pending` to `.used` and syncs the token directory. Signing happens only after that durable consumption. A crash can therefore consume a token without delivering a certificate, but can never reuse it; the operator issues a new token after any ambiguous failure. The joiner verifies the returned certificate's signature against the pinned CA, its public key against the local private key, and its canonical common name, then installs `node.key`, `node.crt`, and `ca.crt` with one synced directory rename. Existing identity directories are never replaced.

4. Transaction descriptor and payload

`Command` is a fixed, canonical little-endian record. A transaction descriptor names the database, random batch ID, predecessor data slot, predecessor chain, result chain, payload SHA-256, payload byte count, transaction count, and WAL frame count. The result chain is the domain-separated hash of the predecessor chain and canonical descriptor fields. It is a log-chain identity, not a hash of the full SQLite file.

Payload format `ZXPL` version 1 contains a fixed header, ordered transaction records, ordered WAL frame metadata, and page images. Transaction records tile the frame range exactly; each ends at a SQLite commit frame. Database identity, page size, counts, byte length, and SHA-256 must agree with the descriptor. Authenticated framing leaves 72 bytes of overhead, so the maximum payload is `64 MiB - 73 bytes`. Larger transactions fail before Paxos append and the speculative image is rebuilt from decided state.

5. Journal

Each epoch journal is `paxos-<configuration-id-hex>.log`. Records contain magic, format version, kind, reserved bytes, monotonic sequence, payload length, canonical payload, and checksum. Protocol writes appear in the exact order emitted by `ReplicatedLog`. The host syncs the journal before confirming durable effects or sending dependent messages.

Recovery truncates only an incomplete final record. Bad magic, version, sequence, checksum, or an invalid interior record is fatal. The node does not vote while its durable prefix or a referenced payload is unavailable.

6. Identity, snapshots, and current pointer

Identity format 2 records node ID, database ID, current configuration ID, and the immutable product role. Format 1 is accepted as `data-voter` and upgraded on the next identity write. A role mismatch is fatal; gateways own no identity. The node ID cannot be reused for another logical member. `CURRENT` names one fully installed snapshot generation. Snapshot manifest format 1 records the database ID, sealed configuration, applied slot, chain hash, and database SHA-256.

Authoritative metadata updates use write temporary, sync file, atomic rename, then sync the parent directory. On restart, `current.db`, its WAL, and SHM are discarded. The image is copied from the verified snapshot and the contiguous committed suffix is replayed. Thus a materialized SQLite file is never accepted as newer evidence than Paxos state.

7. Release limits

1. one serialized writer per database; concurrent endpoints route to that writer and do not create multi-leader writes;
2. one through nine static voters (including witnesses) plus a runtime-sized registry of standbys, read replicas, and gateways; automatic voter replacement is roadmap work;
3. at least one voter must be a campaigning data voter; an all-witness configuration is rejected as permanently unavailable;
4. follower reads require explicit stale consistency; default reads use a fresh exact-ballot quorum fence;
5. at most 1024 statements and 64 MiB of copied input in an explicit transaction;
6. one outstanding idempotent sequence per live session, with a bounded last result; an ambiguous timeout must be retried with the same live session and sequence;
7. no cross-database transaction and no automatic sharding;
8. every production TCP channel uses per-node mTLS; PSK-only TCP requires the explicit numeric-loopback development profile, while plaintext TCP is available solely behind the failpoint-gated test switch;
9. POSIX durability only for this release candidate.

8. Operator upgrade procedure

1. Run `zaxon integrity-check` on every member and take a remote logical backup.
2. Record `version`, database ID, configuration ID, applied slot, and snapshot.
3. Stop one follower, upgrade it, and wait until its applied slot matches.
4. Repeat for the other follower, then the leader. Exact wire-version changes require a documented bridge and must not use this rolling procedure.
5. Run integrity checks and compare logical hashes. On failure stop the node; never delete or rewrite a journal to force it to join.

Disaster recovery begins with copies of all node directories. `zaxon recover --data <dir>` performs the normal journal-authoritative rebuild and integrity check. Interior durable-prefix corruption has no automatic repair command: restore a verified logical backup or obtain an authenticated snapshot from a healthy quorum. Manual gap filling or journal editing is unsupported.