POD 0001: The Paxos Odin Discussion Process

COMMITTED

CATEGORY
Process Memo

STATUS
Committed

AUTHORS
Vikrant Rathore <vikrant@insan.ai>, Paxos Odin Contributors

CREATED
2026-09-16

LAST UPDATED
2026-09-17

LABELS
process, documentation, cli

Abstract

This document defines the **Paxos Odin Discussions (POD)** RFC process, metadata schema, authoring lifecycle, and CLI tooling for `paxos-odin`. Modeled directly after the Zen Discussion Series (ZDS) from `zenfmt`, POD records serve as versioned architectural specifications, protocol derivations, and process memos for consensus engineering.

Status and Implementation Boundary

Committed records may be corrected with a dated update; published records are frozen under the lifecycle defined below. Status describes a design decision, not a claim that every proposed feature ships. The registry and each record must agree.

POD	State	Implementation or evidence boundary
0001	Committed	Active process and Typst editorial policy.
0002	Committed	Implemented core; chunk recovery and canonical membership reviewed.
0003	Committed	Implemented durability gates; host persistence remains an obligation.
0004	Discussion	Leases unimplemented; timing, quorum and restart proof obligations remain open.
0005	Committed	Implemented Odin API; earlier version tables are historical.
0006	Committed	Implemented epoch and release fences; state transfer is host-owned.
0007	Committed	Dated review evidence; historical counts kept separate from latest runs.
0008	Committed	Paper safety argument over code; no machine-checked implementation proof.
0009	Committed	Implemented layout and recovery storage; archived matched profiles and measurements.
0010	Committed	Implemented ownership; bounded best-effort resubmission, not guaranteed delivery.
0011	Committed	Implemented Python SDK: C ABI, typed APIs, packaging wheels, and measured evidence. Optional capabilities rejected by capability bit.

The active specification suite updates contracts in place, retains attributable historical measurements, and links references across records. New proposals must distinguish current behavior, proposed behavior, validation gates, and open questions using the RFC template (`docs/pod/template/rfc-template.typ`). A diagram should explain a boundary or invariant; it need not appear in a process record solely for decoration.

Introduction

Distributed consensus libraries require uncompromising precision. Subtle design choices - such as write-ahead ordering, sliding-window recycling, and quorum intersections - cannot be captured solely in inline source comments or transient issue tracker threads.

POD provides a structured, versioned, Typst-rendered specification pipeline embedded in the repository.

The POD Lifecycle

A POD document progresses through five standardized states:

1. **Prediscussion (Draft)**: A placeholder file named `XXXXX-<slug>.typ` created from `docs/pod/template/rfc-template.typ`. In this state, authors outline ideas and gather initial informal feedback.
2. **Discussion**: A numbered document (`NNNN-<slug>.typ`) actively reviewed by maintainers and contributors.
3. **Committed**: Consensus has been reached, the design is accepted, and implementation is scheduled or completed.
4. **Published**: The specification is finalized and frozen as a permanent reference.
5. **Abandoned**: The proposal was withdrawn or superseded by another record.

Numbering & Promotion Workflow

To prevent Git merge conflicts on sequence numbers across branches, new proposals begin with the placeholder `XXXXX`.

The `paxodin` automation manages the entire lifecycle:

```
# 1. Create a new draft
./bin/paxodin pod new leader-leases

# 2. List all active records and draft placeholders
./bin/paxodin pod list

# 3. Promote the draft to the next permanent 4-digit number
./bin/paxodin pod promote leader-leases

# 4. Compile PDFs via Typst
./bin/paxodin docs pod
```

Registry and Compilation

Every promoted record is registered in `docs/pod/registry.typ`. The document suite is compiled to PDF using the installed `typst` binary:

- Individual records: `docs/build/pod-NNNN-<slug>.pdf`
- Master Index: `docs/build/pod-index.pdf`
- Complete Book: `docs/build/paxos-spec.pdf`

The Zen of Odin for InsanAI

Every POD, and every line of Odin in this repository, is written under one short creed. It is quoted in full so that a reviewer can point at the line a change violates.

> *Data is real; code is just the stream.*
> *Explicit is better than a hidden scheme.*
> *Simple blocks beat abstractions built too high,*
> *A mere mortal should see how the segments tie.*
> *Keep close to the metal, let allocations show,*
> *Pass your contexts cleanly so the lifetimes flow.*
> *Errors are values, never cast aside,*
> *Handle them explicitly; let nothing hide.*
> *Fail with grace, let diagnostics guide:*
> *Show the break, the hint, the fix inside.*
> *Design for speed, let safety lead the pace,*
> *Waste no cycle, leave no leaking trace.*
> *Keep it simple to use, explain, and maintain,*
> *So years from now, the logic remains plain.*
> *Coherence beats purity when real problems strike,*
> *But structure your memory as hardware would like.*

Structural Constraints

The creed is enforced by `tools/check_style.py`, which `make vet`, `make check`, and the `paxodin check` command run before anything else. A hard limit fails the build.

1. File boundary

- **Maximum file length:** a single source file must not exceed 1,408 physical lines, including comments and blank lines. The core protocol is therefore organized into ten files, each with a single responsibility: `ballot.odin`, `bit_set.odin`, `membership.odin`, `ledger.odin` (durable state), `messages.odin`, `effects.odin`, `node.odin` (lifecycle and queries), `election.odin` (phase one), `consensus.odin` (phase two, timers, dispatch), and `ownership.odin` (rotating slot ownership).

2. Line width boundaries

- **Soft limit (99 columns):** lines should be wrapped at or before 99 columns; the checker lists offenders with `--soft`.
- **Hard limit (108 columns):** no line may exceed 108 columns; a longer line fails the build. Tabs count as four columns.

3. Procedure code density

- **Maximum scope (70 lines):** the body of a procedure must not exceed 70 lines of actual execution logic.
- **Exclusions:** blank lines, whitespace-only lines, comment lines, and ornamental divider lines are not counted.

4. Elm-style error handling and diagnostics

- **Actionable reporting:** an error does not merely state what failed; it explains why and provides an actionable path to resolution. In code this is the `Error` enum plus `explain_error`, a data table with one entry per value, and a test that fails when a value has no entry.

- **Diagnostic structure:** every error block or runtime diagnostic carries three parts: the **context** (the failing input or state), the **hint** (the assumption or constraint that was breached), and the **remediation** (how to fix it). The durability gate's `-- DURABILITY ORDER VIOLATION --` banner and every compile-time `#assert` message follow the same shape.

5. Performance and longevity architecture

- **Resource-optimum design:** memory layouts favour mechanical sympathy: contiguous arrays (the `Ledger` columns and bitmaps, the inline `small_array` effect buffers), predictable transformations, and no redundant payload copies into effects. Values are copied into owned ledger storage; borrowed effect pointers have explicit lifetimes.
- **Safety through visibility:** performance never justifies unvetted cleverness. The library relies on Odin's type checking, explicit compile-time bounds (`#assert`, `where` clauses), and the runtime durability gate rather than implicit trust.
- **Long-term maintainability:** every engineering decision must pass the "mere mortal explainability test". An optimisation that cannot be explained simply to a teammate is refactored into a simpler, flatter structure.

Documentation Policy and Editorial Guidance

Typst is the canonical format for project documentation. Markdown is reserved for GitHub-facing entry pages, currently `README.md`, `README.ko.md`, and `CONTRIBUTING.md`. New Markdown documents require a specific GitHub-facing purpose; convenient rendering alone is not a reason to create a second documentation tree.

Keep teaching material in `docs/book/`, design and process records in `docs/pod/records/`, and release notes in `docs/releases/`. Benchmark drivers belong under `bench/`; their explanations belong in the book and their design evidence in POD records. JSON, CSV, patches, and profile archives remain machine-readable evidence under `bench/results/`. Temporary agent notes and scratch plans do not belong in the repository; keep them outside the tracked workspace.

Book editorial guide

The book takes inspiration from Feynman's concrete explanations, Lamport's explicit reasoning, Knuth's integration of programs and exposition, and Dijkstra's economy and precision. These are editorial aims, not quotations, endorsements, or an attempt to imitate an author's voice.

Build understanding in layers

1. Start with a small situation the reader can draw: three voters, one slot, one delayed message. State the question before naming the mechanism.
2. Ask the reader to predict an outcome. Work through the relevant events and say why each is legal. Include a failure case that tests the tempting shortcut.
3. Define the invariant and its assumptions. Separate safety from progress, a protocol fact from a node's knowledge, and the core's duties from the host's.
4. Show a short implementation excerpt next to the obligation it satisfies. Label sketches as sketches. Keep quoted identifiers, field sizes, and indexing current.
5. Ask a transfer question: change a quorum, a crash point, or a storage boundary. Give enough information to reason about it without guessing hidden assumptions.

Reference sections can be direct. Do not force every section into a lesson template or claim that every chapter contains an exercise pattern it does not actually use.

Write precise, readable prose

Use one main claim per paragraph. Prefer concrete subjects and active verbs. Define terms before relying on them; use the same term for the same state throughout. "Chosen", "known to be chosen", "released", and "applied" are distinct events.

Avoid claims such as "obvious", "inevitable", "production-grade", "zero cost", or "always fastest" unless the text supplies the necessary evidence and scope. A counterexample is more useful than a slogan. A paper argument over inspected code is not machine-checked implementation correctness.

Name chapters by subject or use a generated reference; handwritten chapter numbers drift. Keep default validation commands distinct from larger recorded runs. Historical design records and measurements should retain their dates and original context.

Make diagrams do explanatory work

Use native vector diagrams in `docs/book/figures.typ` for protocol and storage reasoning. Each figure needs a question, explicit labels, and a caption explaining its conclusion and assumptions. Colour supplements words; it must not be the only indication of state. Label the direction of time and distinguish data flow from required ordering.

Keep related diagrams near the explanation. Check rendered pages for overlap, clipping, tiny text, misleading arrows, and page breaks. Align quantitative bars at zero. Print units and values; state when panels use different scales. Use HTML frames for diagrams whose layout would otherwise disappear in Typst's experimental HTML export, while keeping the surrounding explanation and tables as text.

Keep measurements attributable

Generate book timing tables and charts from the archived matched JSON and memory charts from the recorded CSVs. State the workload, timing boundary, sample count, compiler policy, and where raw results can be found. Explain uncertainty: passing a regression gate is not proof of no regression.

Distinguish inline storage, allocated heap/stack, and resident process memory. Never sum overlapping metrics. Preserve the historical durability harness separately from the matched CPU suite. Update both READMEs when the leading measurement changes.

Review and build

Run `make docs` to build the book, design records, releases, and HTML. Review the PDF pages containing changed figures and tables; successful compilation alone does not establish legibility. The HTML exporter can warn about unsupported page styling; confirm diagrams survive as SVG frames. Documentation-only edits do not require rerunning the performance experiment, but must not change or relabel its raw evidence.

References

- Lamport, Leslie. "The Part-Time Parliament." ACM TOCS, 1998.
- Zenfmt Monorepo ZDS Architecture (insanai/zenfmt).