Zen Working Group
Request for Discussion and Implementation Record

ZDS 0001
Process Memo

The Zen Discussion Process

STATE
committed

CREATED
2026-08-06

LAST UPDATED
2026-08-08

DISCUSSION
Process document

LABELS
documentation, process

INTENDED STATUS
Committed

AUTHORS
Zen Contributors <team@insan.ai>

Status of This Memo

This document is an internal Zen 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. Abstract

The zenfmt monorepo needs a durable decision-record process for architectural, format, performance, and documentation changes across the conversion library and the `zenfmt` command-line tool. A document converter accumulates decisions that are invisible in the code and expensive to relearn: which parts of a source format are deliberately dropped, why a construct maps to one Markdown spelling rather than another, and what the plugin contract promises a reader that the engine will never do on its behalf. The project uses Zen Discussions, or ZDS, as RFC/RFD-style Typst documents that support structured reasoning, long-lived references, high-quality PDF output, and an HTML discussion website.

This memo defines the lifecycle of a ZDS, the placeholder numbering workflow, the authoring expectations, and the Typst project layout.

2. Introduction

A Zen Discussion is a Typst document stored in git under `docs/zds/records`. Each discussion is part design memo, part review artifact, and part historical record. The structure is intentionally close to IETF RFCs and Oxide RFDs because those formats force explicit scope, status, rationale, alternatives, and operational considerations instead of relying on implicit context.

ZDS is used for topics such as:

- the document intermediate representation and its compatibility rules
- the plugin contract: what a reader may assume, what a writer must handle
- the mapping from a specific source format to the IR, and the constructs that mapping deliberately discards
- performance and allocation budgets, and the benchmarks that defend them
- security decisions, notably the handling of untrusted archive and markup input
- contributor and documentation process decisions

Active design reasoning, trade-offs, and decisions belong in ZDS. The Typst book under `docs/book/`, when it is written, remains the descriptive manual of the current system; a ZDS records why the system became that way.

2.1. Why a converter needs written records

Every format zenfmt reads is a lossy projection onto one shared IR, and every format it writes is a lossy projection back out. The interesting engineering is almost entirely in those two projections, and almost none of it is legible from the code alone. A reader that skips a construct and a reader that has not yet implemented that construct look identical in a diff. Only a record distinguishes them.

This is why a mapping decision — even a small one, such as rendering a single-cell table as a paragraph — belongs in a numbered document with the alternatives that lost.

3. Terminology and Scope

- **ZDS**: a Zen Discussion document
- **placeholder draft**: a local Typst draft using the placeholder number `XXXXX`
- **assigned ZDS**: an accepted discussion with a permanent four-digit number
- **registry**: the Typst metadata list in `docs/zds/registry.typ` used by the index and bundle entry point
- **bundle**: the experimental Typst export target that emits the HTML index, HTML discussion pages, and per-ZDS PDFs from one entry point

This memo defines the repository workflow for ZDS. It does not define future CI automation in full detail, but it does define the expected behavior of that automation.

4. Discussion Lifecycle

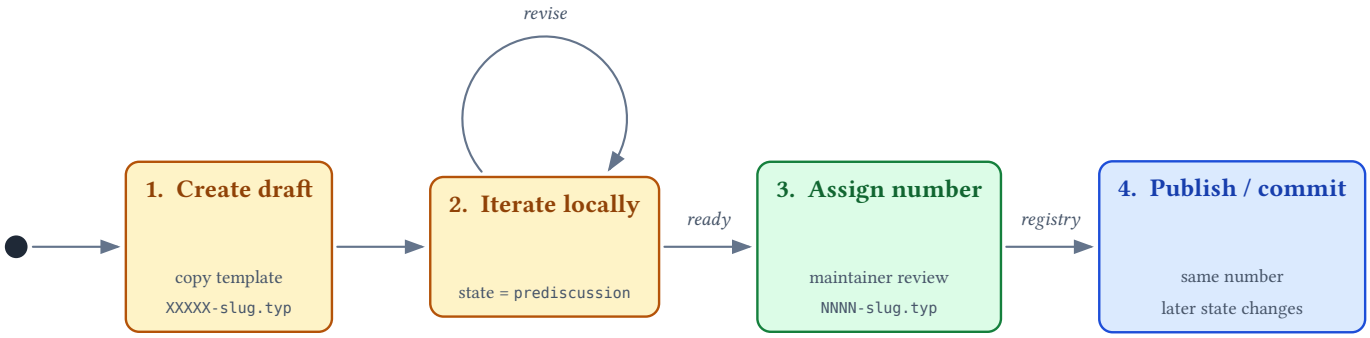
New discussions begin as placeholder drafts.

- Contributors copy `docs/zds/template/rfc-template.typ` to `docs/zds/records/XXXXX-<slug>.typ`.
- The document uses the placeholder number `XXXXX` while the author iterates.

- The registry-driven Zen Discussions index includes placeholder drafts after assigned discussions.
- When the draft is ready, maintainers assign the next permanent ZDS number and rename the file to `NNNN-<slug>.typ`.
- The registry is updated with the document's area, status, summary, source path, HTML path, and PDF path.

This keeps cloning and local authoring simple. Contributors do not need a global lock or remote numbering check just to start writing.

4.1. Local Workflow Diagram



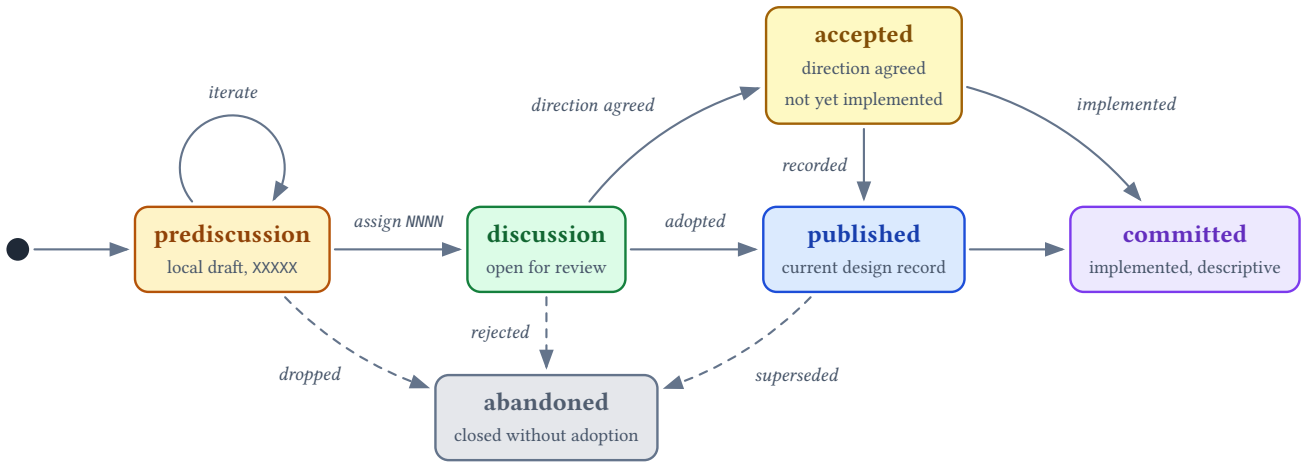
5. States

ZDS documents can move through these states:

- `prediscussion`: local draft or early working document
- `discussion`: open for review and feedback
- `accepted`: accepted as the intended direction, not yet fully implemented
- `published`: accepted into the repository as the current design record
- `committed`: fully implemented and now descriptive of the current system
- `abandoned`: deliberately closed without adoption

5.1. State Transition Diagram

Solid edges are the forward path of a discussion; dashed edges close a document without adoption. The filled dot is the moment a contributor copies the template.



5.2. Transition Detail

Transition	Actor	What happens
start ⟶ `prediscussion`	contributor	Copy `template/rfc-template.typ` to `records/XXXXX-<slug>.typ` and iterate locally under the placeholder number.
`prediscussion` ⟶ `discussion`	maintainer	Run `zig build zds-promote -- <slug>`: it assigns the next four-digit number, renames the file to `NNNN-<slug>.typ`, rewrites the metadata for discussion, and appends the `registry.typ` and `bundle.typ` entries.
`discussion` ⟶ `accepted`	maintainers	Review converges on the proposed direction; implementation has not landed yet. The registry state changes, nothing is renamed.
`discussion` ⟶ `published`	maintainers	The document itself is the deliverable (process memos, assessments, records) and is adopted as the current design record.
`accepted` ⟶ `published`	maintainers	The accepted direction is written up as the repository's current design record while implementation continues.
`accepted` / `published` ⟶ `committed`	maintainers	The described system is fully implemented; the ZDS is now descriptive of the current system rather than a proposal.
any pre-committed state ⟶ `abandoned`	maintainers	The discussion is deliberately closed without adoption: dropped while drafting, rejected in review, or superseded by a later ZDS. The number is never reused.

6. Authoring Rules

ZDS documents SHOULD follow the RFC-style template in `docs/zds/template/rfc-template.typ`.

Each ZDS is expected to include, at minimum:

- an abstract or decision summary
- a clear statement of scope and goals
- the main design or proposal
- considerations for security, operations, or workflow where relevant

- alternatives and unresolved questions
- Keep one ZDS per file under `docs/zds/records`.
- Store document metadata in the `#let zds-*` assignments at the top of the file.
- Update `docs/zds/registry.typ` whenever a ZDS is added, renumbered, renamed, or changes lifecycle state.
- Build through `zig build` once Zig is installed; Typst remains the document compiler.

The ZDS Typst layer is allowed to use external Typst packages for diagrams, charts, richer tables, or specialized layout when those packages improve the clarity of the PDF output. The repository should prefer existing Typst packages over building a custom extension framework here.

6.1. Format Records

A ZDS that adds or changes a format plugin carries three sections beyond the template, because these are the parts of a converter that reviewers cannot reconstruct from the diff:

- **Mapping table**: every source construct the plugin recognizes, and the IR node it produces. One row per construct.
- **Deliberate omissions**: source constructs the plugin sees and drops on purpose, each with the reason. This is the section that distinguishes a decision from a gap, and it is the section reviewers should read first.
- **Round-trip expectations**: what survives a conversion to the target format and back, and what does not. A converter that claims fidelity it does not have is worse than one that documents its losses.

Two rules follow from the shared IR. A change to `Block`, `Inline`, or their tags affects every plugin at once and requires its own ZDS. A change confined to one plugin's mapping does not, and may be recorded as an amendment to that plugin's existing record.

7. Typst Project Layout

The ZDS tree is a first-class Typst project:

- `docs/zds/records/` contains standalone ZDS source files.
- `docs/zds/template/rfc-template.typ` is the starting point for new ZDS drafts.
- `docs/zds/registry.typ` is the metadata registry used by the index and bundle.
- `docs/zds/index.typ` renders the PDF/HTML index from the registry.
- `docs/zds/bundle.typ` uses Typst bundle export to emit `index.html`, one HTML page per ZDS, and one PDF per ZDS.
- `docs/shared/zds.typ` and `docs/shared/theme.typ` provide the shared document frame, styling, and index components.
- `docs/book/` and `docs/book.typ` hold the skeleton of the future zenfmt book. The theme, callouts, and figure helpers are in place; the chapters are not written yet, and no build step compiles them.

Typst bundle export and HTML export are experimental in Typst 0.15. They are useful for the ZDS website, but the repository should still keep standalone PDF compilation for each ZDS as the stable archival path.

8. Build Integration

The root `build.zig` owns the ZDS build steps. Records are discovered by scanning `docs/zds/records`, so adding a record never requires a build-file edit:

- `zig build zds` compiles every numbered record to `docs/build/zds-NNNN-<slug>.pdf`.
- `zig build zds -Dzds=<number-or-slug>` compiles a single record; the number may be unpadded (`-Dzds=2`), and a placeholder draft can be selected by its slug for proofreading before promotion.
- `zig build zds-index` compiles the registry-driven index to `docs/build/zds-index.pdf`.
- `zig build zds-site` compiles the experimental HTML bundle to `docs/build/zds-site/`.

Lifecycle management is owned by `tools/zds.zig`:

- `zig build zds-list` prints registry entries and placeholder drafts, and warns when a record file and the registry disagree.
- `zig build zds-new -- <slug>` creates `records/XXXXX-<slug>.typ` from the template with today's date.
- `zig build zds-promote -- <slug>` performs the `prediscussion` → `discussion` transition described above.

Direct Typst commands are useful while editing:

```
typst compile --root docs docs/zds/records/0001-zds-process.typ \
  docs/build/zds-0001-zds-process.pdf
typst compile --features html,bundle --root docs --format bundle \
  docs/zds/bundle.typ docs/build/zds-site
```

9. Number Assignment and CI

The repository supports a local numbering workflow for maintainers who manage discussions directly in git history. The `tools/zds.zig` command, wired into the build as `zds-list`, `zds-new`, and `zds-promote`, automates it:

- list current ZDS entries and placeholder drafts
- assign the next permanent four-digit ZDS number to a placeholder draft
- rename the file from `XXXXX-<slug>.typ` to `NNNN-<slug>.typ`
- rewrite the `zds-number` metadata and transition the document into discussion
- append the metadata entry to `docs/zds/registry.typ` and the export blocks to `docs/zds/bundle.typ`

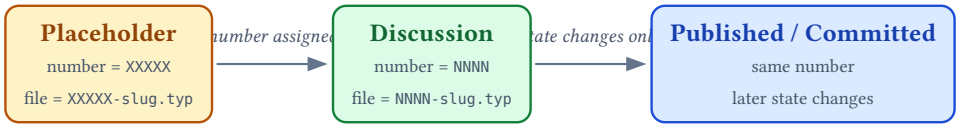
The tool performs ordinary file edits reviewed in the same change as the discussion — there is no hidden state, and every step can still be done by hand. The generated registry summary and area come from the draft's `zds-discussion` and first `zds-labels` entry; review both before committing.

That local flow is sufficient for small teams and direct-maintainer repositories. A future CI workflow can still:

- detect placeholder ZDS files that are ready for discussion
- reserve the next permanent ZDS number
- rename the file from `XXXXX-<slug>.typ` to `NNNN-<slug>.typ`
- rewrite the `zds-number` metadata

Both flows preserve a stable numbered sequence in shared history. Teams can choose local CLI assignment, CI assignment, or a combination where CI validates numbering but maintainers assign numbers intentionally.

9.1. Numbering State Diagram



10. Security Considerations

An ambiguous discussion process creates implementation drift and undocumented design assumptions. For a converter this is not cosmetic. zenfmt parses untrusted input by construction: its whole purpose is to accept a file someone else produced. Decisions about archive expansion limits, entry-count and decompression-ratio caps, XML entity expansion, and path traversal inside container formats are security decisions, and they must stay traceable to an explicit record rather than living as an unexplained constant in a parser.

Requiring explicit sections for scope, considerations, and alternatives reduces the chance that such limits are relaxed later by someone who cannot see why they were chosen.

11. References

- `IETF RFCs` for explicit memo structure, status, and considerations
- `Oxide RFDs` for engineering discussion records in a source repository
- `Typst bundle export` for the multi-file ZDS website and per-ZDS PDFs
- ZDS 0002, the zenfmt architecture and implementation record