
The Guidedog Discussion Process

Design, rationale, and review

01 / SOURCE

Write with meaning.

02 / STRUCTURE

Make relations clear.

03 / PUBLICATION

Read on screen and paper.

Contents

Abstract	1
Introduction and scope	1
Terminology	2
Discussion lifecycle	2
Metadata and record structure	4
RST project and build workflow	6
Number assignment and validation	7
Guidedog CLI management contract	7
Security and operational considerations	9
Backwards compatibility and migration	10
Alternatives considered	10
Acceptance criteria and unresolved questions	10
Review history	10
References	11

Number: 0001
Title: The Guidedog Discussion Process
State: committed
Type: Process
Authors: Vikrant Rathore
Created: 2026-09-29
Updated: 2026-10-11
Discussion: Local review; no external discussion URL assigned
Labels: process

Abstract

Guidedog Discussions (GDS) are the project's RFC/RFD-style specification and decision records. The subject is Guidedog, a Sphinx-like documentation system built in Odin. This revision adapts the lifecycle from ZDS 0001 and preserves its lifecycle and numbering process, and adds explicit PEP-inspired guidance for rationale, compatibility, review history, and implementation evidence. GDS is a collection of standalone discussions, not a book divided into chapters.

Introduction and scope

Guidedog builds a documentation project into a responsive HTML site or a PDF book using Typst. Its scope includes navigation, cross-document references, directives and roles, themes, project configuration, incremental builds, local preview, and single-binary distribution. Guidedoc is one subsystem: its conversion library. GDS decisions govern the whole product, not a renamed zenfmt file converter.

A discussion explains one coherent proposal and the reasoning that makes it worth adopting. It can specify a feature, a format mapping, a process, or an assessment. A future user manual belongs outside GDS. The HTML index connects standalone records; each record has its own PDF. There is no PDF index or combined PDF build.

The process retains ZDS 0001's local placeholder drafts, maintainer-assigned numbers, state transitions, registry, standalone reStructuredText records, and format-record obligations. RFCs inform memo structure and considerations; Oxide RFDs inform repository-based discussion; PEPs inform focused proposals, rationale, alternatives, and the distinction between editorial readiness and technical acceptance. We adopt these practices without claiming the governance or authority of those projects.

This process is implemented by `guidedog gds` for reStructuredText records. The register and lifecycle events record the maintainer-requested reconciliation of implementation status. The inherited process alone is not acceptance evidence. Its original wording remains in `legacy/0001-zds-process.rst`.

When a Guidedog discussion is needed

Write a GDS for user-visible command or configuration changes; document graph and reference semantics; RST directives, roles, CommonMark extensions,

or Typst behavior; HTML and PDF builders; theme contracts; plugins and trust boundaries; cache invalidation; memory and CPU limits; packaging; and project governance. A proposal explains effects on existing documentation projects and on migration from Sphinx where relevant. State whether a Sphinx feature is supported, adapted, or out of scope. Sphinx compatibility is not inferred from RST support alone.

Bug fixes preserving specified behavior, routine refactors, and additional tests usually amend an existing record or need no new record. The future user manual explains how to use Guidedog; release notes state what shipped; GDS explains what was decided and why. Do not move normative design decisions into manual chapters.

Terminology

A **placeholder draft** has number `XXXXX` and state `prediscussion`. An *assigned GDS** has a permanent four-digit number. The **author** maintains the proposal and records feedback. A **maintainer** assigns numbers and records project decisions. The **registry** describes available discussions and historical references. A **format record** specifies parsing, mappings, policy refusals, and fidelity.

Discussion lifecycle

New proposals begin as `records/XXXXX-<slug>.rst`. Copy the template, fill in metadata, and iterate locally. Placeholder drafts appear after numbered records in the index. When ready for review, a maintainer assigns the next unreserved number, renames the file, and updates metadata, registry, and bundle together. A number is an identifier, not evidence of approval. It is never reused.

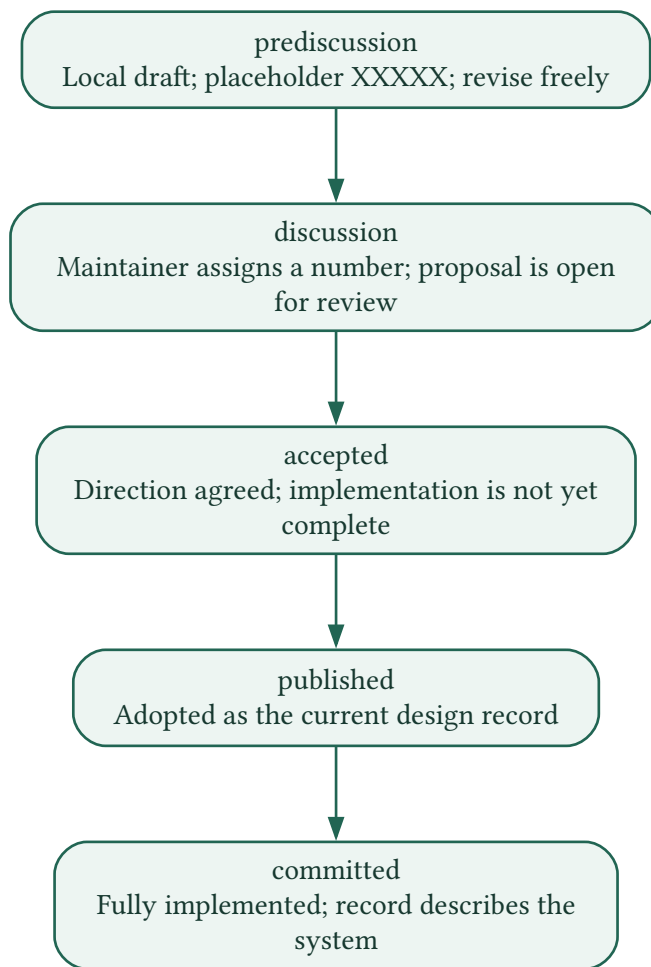


Fig. 2 The usual forward path. The transition table defines shortcuts and abandonment.

Transition	Meaning and responsible actor
prediscussion → discussion	Maintainer assigns the number and registers the record.
discussion → accepted	Maintainers agree on the proposed direction.
discussion → published	Maintainers adopt a document whose text is the deliverable.
accepted → published	Maintainers adopt the design record while implementation continues.
accepted / published → committed	Maintainers verify implementation evidence.
Any pre-committed state → abandoned	Maintainers record why the proposal was closed.

These are the six inherited states. Do not replace them with PEP states such as Final, Active, or Rejected. A Git commit does not mean the GDS state is `committed`. A successful Typst compile does not mean the design is accepted. Preserve

numbered abandoned records. For a committed design superseded later, retain its history and link a successor instead of pretending it was never implemented.

Metadata and record structure

Each file starts with one `:gds-name:` metadata field per line. Required names are `number`, `title`, `state`, `type`, `authors`, `created`, `updated`, `discussion`, and `labels`. Scalar values are plain text. Authors and labels are JSON arrays of strings. The block is data; it is never evaluated as code. Published headers use `Number`, `Title`, `State`, `Type`, `Authors`, `Created`, `Updated`, `Discussion`, and `Labels`. Authors and labels appear as readable lists. This presentation does not rename the source fields or change what the GDS CLI reads and edits. The type is `Standards Track`, `Process`, or `Informational`; it does not change the inherited state machine. Dates use ISO format. An unavailable discussion link is stated plainly; never fabricate an issue, author contact, or acceptance decision.

An assigned number must match its filename. A placeholder is identified by its slug, so multiple `xxxxx` drafts do not collide. Related, superseding, and superseded records are named explicitly in prose and registry metadata when relevant.

A record normally includes abstract, motivation, scope, specification, rationale, security and operational considerations, backwards compatibility, alternatives, open questions, reference implementation or implementation status, acceptance criteria, review history, and references. Headings may fit the subject. A section that does not apply says why; avoid boilerplate that adds no information.

The specification states observable behavior and testable invariants. Rationale explains choices; it is not a second ambiguous specification. Examples illustrate the contract and do not silently weaken it. If uppercase `MUST`, `SHOULD`, or `MAY` is used normatively, cite RFC 2119 and RFC 8174 and explain their use. Ordinary precise English is preferred where no formal requirement keyword is needed.

One discussion lives in one `reStructuredText` source file. Shared templates are allowed. Normative prose stays in the record. Graphviz diagrams and math directives are ordinary source content, so the project engine can check and render them. Split an oversized proposal into focused discussions with explicit dependencies. The template is a starting point rather than a demand for a particular narrative order.

Format records

Retain the three ZDS 0001 requirements: a mapping table for each recognized source construct and its semantic representation; deliberate omissions with reasons; and round-trip expectations. Because this project currently has only HTML and PDF outputs, round-trip recovery may be unavailable. Say so rather than implying that PDF can reconstruct source. Distinguish parsing coverage, target representation, policy refusal, intentional omission, and unfinished work.

Full RST requires a coverage ledger for the specification, standard directives, and roles. CommonMark names the pinned specification and suite. Typst behavior names the pinned toolchain. Every mapping change updates its fixtures and record. Changes to shared AST semantics require a dedicated GDS; localized mapping corrections may amend the existing format record.

Coding and writing rules

Every new source file has at most **1,408 physical lines**. A line should remain within **99 characters** and must not exceed **108 characters**. Source is Odin and every other implementation language shipped with it: CSS, JavaScript, HTML templates, shell, Rust, C, and TOML. Documentation (Markdown, reStructured-Text, and Typst prose, diagrams, and markup structure, these records included) is written for readers and exempt from every one of these limits: line width, file length, and procedure length apply to code only. Every procedure has at most **70 lines of logic**, excluding its signature, blank lines, comments, and ornamental delimiters. Multi-statement lines cannot be used to evade the rule. Split work by responsibility, not by arbitrary line count.

`odin run tools/check` uses tokens to distinguish Odin code from strings and comments. It includes procedure literals and multiline expressions. The current checker fails both soft-width findings and hard limits. Generated code must be split or redesigned to fit; it has no blanket exemption. Inherited Zen records remain an explicitly historical archive and are not presented as compliant new source.

Prefer small values, distinct IDs, exhaustive tag switches, checked arithmetic, explicit limits, and narrow package imports. Avoid unchecked pointer casts, disabling bounds checks, recursion driven by input, and thread-shared mutation. Errors due to user data return results. Assertions state internal invariants that have already been established by validation.

Memory safety

Guidedog must never crash on its input or on a failed allocation. Odin gives the tools; these rules say which ones every change uses. Tests enforce the rules that a scan can check (`tests/memory_safety_test.odin`); review enforces the rest.

- **Check every allocation.** `make`, `new`, `clone`, `aprintf` and the rest return an `mem.Allocator_Error` or an `ok`. A caller takes the value and the result together (`(x, err := make([T], n)`), stops with `or_return`, or chooses a fallback with `or_else`, and never indexes, dereferences, or passes on a value whose allocation failed. Work that cannot check each allocation (a library that goes on with what it has) allocates through a `host.Guard`: the guard remembers the first refusal and refuses every later allocation, so the work only shrinks, and the caller fails the whole step with `host.memory` when the work ends. Arenas (`host.Arena`, `runtime.Arena`, Jinja's own) stay consistent when their backing allocator refuses a block; `mem.Dynamic_Arena` does not, and is not used in product code (`tests/memory_safety_test.odin` checks). Code that cannot survive a refusal, such as Odin's parser, runs on memory the caller's allocator never refuses (`odindoc` parses in virtual memory of its own) within a bound reserved ahead. Tests refuse every allocation of a run in turn (`tests/allocation_failure_test.odin` and the libraries' `memory_test.odin` and `refusal_test.odin`).
- **Bounds checks stay on.** No `#no_bounds_check` and no `-no-bounds-check`, in code, build commands, scripts, or documentation. A loop that needs a check elided slices once before it, so the compiler proves the bound.
- **No pointer arithmetic or multi-pointers outside C bindings.** `[^]T`, `uintptr` arithmetic, and `transmute` between unrelated types belong to the

bindings of Graphviz, tree-sitter, the Typst bridge, and curl, which pass C’s memory to Odin as slices at their boundary. Elsewhere a region is carved by slicing the bytes first (bounds-checked) and then reinterpreting the slice (`slice.reinterpret`); the offset of a slice within its text comes from `core.offset_in`, which checks that it lies inside. Reading a frame’s address to measure the stack, or a native `#soa` slice’s header, is allowed with a comment saying so. Callback user data is the one `rawptr` cast allowed: the callee casts back to the type its registrant passed.

- **No views past their memory.** A string or slice into an arena, a session, or the temporary allocator is used only while that memory lives; what outlives a step is copied into the caller’s allocator before the step’s arena is freed (as `scratch_close` copies reports and problems). Library code never calls `free_all(context.temp_allocator)`.
- **Nil-checked pointers.** A procedure that finds something returns `^T` and `nil`, or `(T, bool)`, or `Maybe(T)`; its callers check before dereferencing. Tables that point into a struct use typed pointers (a union of them where kinds differ), never `rawptr`.
- **Ownership is explicit.** Each entry point owns an arena of its own, freed by one procedure (`unload`, `release`, `destroy_environment`) with `defer` at the call site.
- **AddressSanitizer.** `tools/sanitize.sh` runs every suite and the end-to-end tests with `-sanitize:address` before a release, and after changes to memory handling.

Write documentation as an explanation addressed to a careful colleague. Define a term before relying on it. State an invariant beside the algorithm that needs it. Give a small example, then show where the example stops applying. Explain why a choice exists and which observable behavior depends on it. Use plain English and precise claims, following the requested spirit of Feynman, Knuth, Lamport, and Dijkstra without imitating their prose or inventing quotations.

A public procedure’s comment states what it does, required preconditions, ownership, view lifetime, failure behavior, and any complexity bound that has been justified. A format record has a mapping table, conformance ledger, target limitations, policy refusals, and regression fixtures. “Unsupported” must identify a specific construct.

RST project and build workflow

The discussions are a separate Guidedog project. The manual is another project. Both begin with `guidedog quickstart`. A record’s prose belongs in `records/.template/rfc-template.rst` supplies new drafts. `registry.rst` lists identities and states. `bundle.rst` supplies the reading order. The CLI refreshes their managed regions without discarding surrounding prose.

```
guidedog gds index
guidedog gds check
guidedog gds build --target=both
guidedog gds build guidedoc-odin --target=pdf
guidedog build html docs/gds -W
```

RST publication uses `lib/project` and Guidedoc, like any other project. HTML is a coherent site in `docs/gds/_build/html`. A selector limits PDFs, not the linked HTML graph. PDF builds produce one book per active record in `docs/gds/_build/pdf`. There is no combined design book by default. Typst is required for PDF, not for RST HTML.

The HTML template supplies navigation, search, and a page outline. The default is light. Readers may choose dark or system appearance. PDF uses the project book template. Graphviz diagrams render to static SVG assets. A typesetting template can remain Typst; the authored record is RST.

Legacy Typst catalogs remain supported by the library for existing callers. They use their original publication route. A single RST project must not mix native Typst records into its active record catalog.

Number assignment and validation

Guidedog has its own number sequence. GDS 0001 defines this process; the next assignment was 0002 for the CLI. GDS 0003 is the engine record. Records 0004 through 0006 cover budgets, the Odin domain, and projects. The next available number is 0007. Inherited zenfmt records do not reserve Guidedog numbers. New drafts retain `xxxxx` until a maintainer promotes them. Promotion chooses the next number after existing Guidedog records, explicit Guidedog reservations, and local `gds/NNNN` branches.

Validation checks unique assigned numbers and slugs, legal states, metadata and registry agreement, real source paths, and compilable active entry points. A record marked historical may link to the sibling repository but must not be included in the active PDF build unless its dependencies are present. Promotion updates the filename, metadata, registry, and bundle together. The current CLI does not rewrite prose links: the author reviews and repairs inbound references as part of promotion.

Guidedog CLI management contract

The implementation reference is `../docodin/cmd/docodin/dod.odin`, its DOD process, and its CLI reference. That implementation provides `new`, `list`, `index`, and `check`. It assigns permanent numbers at creation and includes an `ideation` state. Guidedog deliberately retains ZDS 0001's `xxxxx` drafts and six states instead; `promote` and `state` add explicit management of that inherited workflow.

```
guidedog gds new "Incremental project builds" --labels=architecture,build
guidedog gds list --state=prediscussion --label=build
guidedog gds show incremental-project-builds
guidedog gds promote incremental-project-builds --dry-run
guidedog gds promote incremental-project-builds
guidedog gds state 0002 --to=accepted --reason="Review converged"
guidedog gds index
guidedog gds check
guidedog gds build --record=0002 --target=pdf
```

All commands accept `--dir=DIR`, defaulting to `docs/gds`, and `--help`. Diagnostics use the standard Elm-style text or JSON mode and terminal color policy. `list`

and `show` support `--format=table|json`; JSON contains metadata fields and source paths, never terminal decoration. Read-only commands never rewrite files.

Command	Required behavior
<code>new</code>	Create <code>records/XXXXX-slug.rst</code> from the local template. Set date, labels, title, author, and prediscussion. Refuse an existing slug.
<code>list</code>	Sort assigned records numerically, then drafts by slug. Filter by state and label; show historical records only with <code>--include-historical</code>.
<code>show</code>	Resolve number, exact slug, full title, or filename and print metadata and path. Ambiguous identifiers return candidate names without choosing one.
<code>promote</code>	Validate a draft, allocate the next unreserved number, rename it, set discussion, and update registry and bundle references.
<code>state</code>	Apply only a legal transition; require a reason, append review history, and update registry metadata. It never performs a Git commit or publication.
<code>index</code>	Rebuild derived registry entries and bundle includes deterministically; preserve surrounding prose and explicit GDS reservations. <code>--check</code> reports drift only.
<code>check</code>	Validate metadata, unique identities, paths, dates, labels, states, required sections and index/bundle agreement. Prose links need review.
<code>build</code>	Render selected records or all active records to HTML and individual PDFs. HTML builds refresh the complete catalog index; PDF builds create no index.
<code>recover</code>	With <code>--rollback</code> , restore journaled originals unless later edits conflict.

`new` accepts `--author`, `--slug`, and `--labels`. If author is absent, use configured Git identity; if absent, request `--author` with a concrete diagnostic. Do not invent an identity or change Git configuration. Slugs are normalized safe basenames; reject empty names, separators, and path traversal. Labels come from an explicit project

catalog: architecture, parser, renderer, cli, build, theme, typst, library, documentation, process, security, compatibility, performance, and release. Performance labels do not authorize premature optimization.

`promote` accepts an optional `--number=N` for maintainers and checks that it is unreserved in Guidedog records, explicit reservations, and local `gds/NNNN` branches when Git metadata is available. It never fetches remotely. Refuse numbers outside 0001 through 9999, duplicates, and previously assigned GDS numbers. `--discussion=URL` records a real discussion URL when supplied; otherwise leave its absence explicit. Number assignment is not design acceptance and does not automatically open a pull request.

`state` requires `--reason`; moving to `committed` also requires `--evidence` naming implementation and validation artifacts. This records a maintainer's decision, not an inference by the program that code is correct. No `--force` bypass of the state graph is provided. Supersession of a committed record uses a successor reference and leaves the historical state intact.

Mutation commands support `--dry-run` to show paths and metadata changes. They lock the GDS root, reread identities under the lock, stage changes, validate them, and commit with a recovery journal. Multiple file renames are not called atomic. A failed or interrupted operation must roll back or leave a journal that the next command reports with `guidedog gds recover --rollback`. Recovery verifies recorded original and intended contents and refuses to overwrite later edits. A live competing lock reports that the directory is busy. Kernel locks release on process exit; the persistent lock file is not itself a stale lock and is not removed.

The metadata parser recognizes a declarative RST field-list preamble; it must not execute arbitrary Typst to list or promote records. Unsupported dynamic metadata receives a direction to use literal fields. Rewrites touch that preamble and managed registry/bundle regions, preserving the record's prose. Escape strings as Typst literals. Mutating commands never overwrite an existing draft or output unless that operation's contract explicitly allows regeneration.

`check` is read-only. An optional `--render` additionally compiles active records; metadata checks must work without a Typst backend. Exit codes follow the general CLI contract: 0 success, 1 invalid records or drift, 2 usage, 3 resource limits, 4 host/backend failures, 5 internal errors, and 130 cancellation. `list` with no matches succeeds with an empty result. Diagnostics always identify the record, problem, consequence, and exact repair.

Acceptance tests cover new-record defaults, slug collisions, concurrent promotion, reserved numbers, legal and illegal transitions, idempotent indexing, malformed metadata, JSON output, dry-run with no writes, interrupted updates, and recovery that preserves later edits. Run them in temporary directories, never against the repository's actual discussion history. CLI management may use host allocations; that does not weaken Guidedoc's zero-allocation conversion boundary.

Security and operational considerations

Architectural decisions about parsing, resource limits, and publication must be traceable. Ambiguous status or invented implementation claims can cause

unsafe assumptions in code. Review must separate implemented evidence from proposals and identify the owner of every resource boundary.

A record's metadata, examples, and links are reviewed along with its specification. Diagram dependencies must be pinned if external packages are used. Current diagrams use Graphviz directives. Guidedog renders vector output without a downloaded Typst package. `guidedog gds check --render` compiles the active records; `tools/check` checks implementation sources. Documentation prose is exempt from the source width and procedure limits.

Backwards compatibility and migration

GDS is the new name for the inherited ZDS process in this project. Preserve existing numbers and historical meaning. Rename active metadata and paths from `zds` to `gds`. Keep original copied Zen artifacts in the archive where needed. GDS 0001 is the adaptation of ZDS 0001, not a replacement lifecycle invented for this repository.

The old architecture remains historical. New Odin proposals do not silently amend zenfmt's implementation. No chapter directory forms part of the GDS structure.

Alternatives considered

A book would make individual proposals harder to discuss and assign stable status. A new PEP-style state machine would conflict with the existing ZDS process. Automatic numbering at draft creation would lose the local placeholder workflow. These options are not adopted. A future manual may still explain accepted behavior separately.

Acceptance criteria and unresolved questions

The implemented process preserves the inherited six states and permanent numbering. The CLI checks metadata and required sections, regenerates the registry and reading order, and uses locked, recoverable transactions for promotion and state changes. Integration tests cover legal and illegal transitions, reserved numbers, dry runs, concurrent edits, interrupted journals, and recovery that preserves later edits. The actual RST template is exercised by `tests/manual_test.odin`. All six active records are built separately as HTML and PDF during this review. The external discussion venue remains unassigned; no external approval is claimed. GDS 0002 retains the broader CLI obligations not completed for the beta.

Review history

29 September 2026: rename Zen Discussions to Guidedog Discussions; preserve the lifecycle; replace unavailable Zig build instructions with actual Typst commands; incorporate RFC, Oxide RFD, and PEP structural guidance; remove chapter-based drafts.

29 September 2026, tooling review: replace temporary direct-build instructions with the Odin CLI. Record HTML catalog and individual PDF behavior, independent Guidedog numbering, snapshot-based recovery, and manual review of prose references. Keep the existing discussion state; no lifecycle acceptance is inferred from tests.

30 September 2026: add the memory-safety rules to the coding rules: checked allocations, `host.Guard` for work that does not stop at a failed allocation, bounds checks always on, no pointer arithmetic outside C bindings, no views past their memory, nil-checked pointers, and AddressSanitizer runs.

References

- Docodin `cmd/docodin/dod.odin`, `docs/dod/0001-dod-process.rst`, and CLI reference.
- Original ZDS 0001, preserved at `legacy/0001-zds-process.rst` and in `../zenfmt`.
- RFC 7322: RFC Style Guide.
- RFC 2119 and RFC 8174: requirement keywords.
- Oxide RFD 1: Requests for Discussion.
- PEP 1: PEP Purpose and Guidelines.
- PEP 12: Sample PEP Template.

Source-format revision — 2026-10-01

The authored records, catalog, and creation template are now RST. The CLI retains the six-state lifecycle and recoverable metadata transactions. Publication uses the project engine. This is an editorial and toolchain revision; it does not change this record's review state or assert acceptance.

Lifecycle event

2026-10-11: discussion → published. Maintainer-requested reconciliation: the RST workflow and inherited lifecycle are implemented and reviewed.

Lifecycle event

2026-10-11: published → committed. RST discussion management and recoverable lifecycle transactions implemented. `tests/manual_test.odin`; workflow, concurrency, recovery and date tests; native GDS strict HTML/PDF verification; `docs/manual/evidence/beta-review-20261011.md`.