
Guidedog: CLI and documentation toolchain

Design, rationale, and review

01 / SOURCE

Write with meaning.

02 / STRUCTURE

Make relations clear.

03 / PUBLICATION

Read on screen and paper.

Guidedog: CLI and documentation toolchain

Version 0.2.0

© 2026, Vikrant Rathore

Oct 11, 2026

Contents

Abstract	1
Motivation and scope	1
Implemented specification: discussion CLI	1
Specification: conversion command-line behavior	2
Guidedog: the project builder	5
Security considerations	11
Backwards compatibility	11
Alternatives considered	11
Discussion management	11
Acceptance criteria and reference implementation	12
Decisions still requiring implementation evidence	13
Review history	13
References	13

Number: 0002
Title: Guidedog: CLI and documentation toolchain
State: published
Type: Standards Track
Authors: Vikrant Rathore
Created: 2026-09-29
Updated: 2026-10-11
Discussion: Local review; no external discussion URL assigned
Labels: cli

Abstract

This record specifies the Guidedog command line: discussion management, single-document conversion, and the documentation project builder. All three are implemented. The executable converts reStructuredText, CommonMark, and native Typst to HTML and PDF, builds a responsive website and a PDF book from a project, and embeds the Typst 0.15.1 compiler when built with its bridge. Typst packages and system fonts are supported. Open items, such as worker isolation for hostile Typst and a cross-document symbol index, are listed at the end.

Motivation and scope

Users need one command that makes output predictably and explains failures in terms they can act on. The CLI should share engine semantics with embedded callers, while owning terminal behavior, files, resources, Typst, and publication. This record separates what is implemented from what remains open; an implemented command does not by itself establish that every acceptance gate below is met.

Implemented specification: discussion CLI

`lib/cli` owns arguments, validation, terminal presentation, and dispatch. `lib/gds` owns record metadata, catalog selection, lifecycle plans, and recovery. RST publication delegates to `lib/project`. Guidedoc reads and renders the prose. Typst sets PDF pages. RST HTML does not require a Typst installation.

```
guidedog gds new "Cache policy" --author="Your name" --labels=build
guidedog gds list --state=prediscussion
guidedog gds show cache-policy --format=json
guidedog gds promote cache-policy --dry-run
guidedog gds index
guidedog gds check
guidedog gds build --target=both
guidedog gds build guidedoc-odin --target=pdf
```

A selector is a number, exact slug, full title, or source filename. Ambiguous selectors return candidates. `--dir` selects the discussion project. The default remains `docs/gds`.

Metadata is an initial RST field list. Scalar values are plain text. Authors and labels are JSON string arrays. Listing and editing never execute code. The

local RST template supplies new drafts. An author comes from `--author` or Git's configured name. Missing authors receive a correction direction.

Mutations are separate from builds. A mutation holds the kernel lock while it plans and applies changes. The recovery journal records original and intended text. Rollback refuses to overwrite an edit made after the interrupted operation. Closing the lock handle releases the lock. A leftover lock filename is not a busy lock.

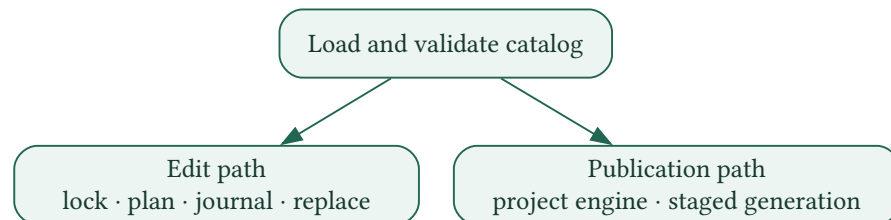


Fig. 3 Metadata edits and publication use different transactions.

HTML builds publish a complete site in the discussion project's `_build/html`. The catalog, search, labels, and cross-document links use the normal project graph. A selected build still refreshes that coherent site. Selected PDFs use a separate generation in `_build/selected/<artifact>/pdf`; they do not replace the complete catalog in `_build/pdf`. PDF builds create one book per selected active record in `_build/pdf`. No selector means all active records. There is no combined PDF by default. A normal `guidedog build html docs/gds` or `build pdf` works too.

The project engine owns cache validity and generation publication. Compiler or renderer failure preserves the previous target generation. Each HTML or PDF target commits independently; both is not a transaction covering both target directories.

Source and metadata limits remain distinct from compiler costs. A record is limited to 1 MiB. Directory entries are limited to 4,096. The recovery journal is limited to 32 MiB. Documentation prose is exempt from source formatting limits. The external Typst compiler's allocations are not bounded by the host's working-memory budget.

Legacy Typst catalogs keep their old metadata and publication contracts. This RST project has no need for Typst's experimental HTML exporter. The source-format change does not promote this record to an accepted state.

Specification: conversion command-line behavior

The first executable is named `guidedog`; `convert` exposes Guidedoc before the project builder exists. This avoids shipping a second CLI name that later needs renaming. `convert` reports each reader's conformance through `formats`: CommonMark passes all 652 examples of 0.31.2; the reStructuredText reader is partial, with its gaps listed in the readers' coverage ledgers.

```

guidedog --help
guidedog --version
guidedog formats
guidedog convert guide.rst --to html --output guide.html
guidedog convert guide.md --to pdf --output guide.pdf
guidedog convert guide.typ --to pdf --output guide.pdf
guidedog convert - --from commonmark --to html --output -

```

`.md` selects CommonMark. `.rst` selects reStructuredText. `.typ` selects native Typst. An explicit `--from` overrides suffix selection; ambiguous or missing selection produces a direction to specify it. No heuristic claims that arbitrary plain text is reliably distinguishable as RST or Markdown. `--to` accepts only `html` or `pdf`; if omitted, the output suffix may select the target. Intermediate Typst is available through `--emit-typst path`, not as a third product target.

`formats` reports actual adapter availability, conformance status, and toolchain version. It must distinguish planned, partial, and conformant readers. Global flags include `--color=auto|always|never`, `--diagnostics=text|json`, and `--quiet`. Use `--` to terminate option parsing. Never evaluate a shell command from `argv`.

ANSI belongs at the terminal boundary

Use the installed Odin package `core:terminal/ansi`. Its constants are escape-code components, so a colored heading combines `ansi.CSI`, a color, and `ansi.SGR`. The terminal renderer is the only package allowed to emit these sequences.

```

import "core:terminal/ansi"

error_style :: ansi.CSI + ansi.FG_RED + ansi.SGR
reset_style :: ansi.CSI + ansi.RESET + ansi.SGR

```

Automatic color requires a terminal on the diagnostics stream, no `NO_COLOR`, and a terminal other than `dumb`. Explicit `--color` takes precedence. JSON and artifact bytes never contain ANSI styling. Diagnostic source excerpts escape terminal control characters, including embedded escape sequences from filenames or source text. Honor redirected stderr, narrow terminals, tabs, combining marks, and Unicode. Color supplements words and markers; it never carries meaning by itself.

`stdout` contains requested artifact bytes or machine-readable command output. `stderr` contains diagnostics. In JSON mode `stderr` is a versioned JSON-lines report stream. Help and version use `stdout` on success. Version output includes compiler, Guidedoc schema, and Typst backend versions when applicable.

Exit	Meaning
0	Completed successfully; warnings may exist
1	Invalid document, unresolved references, or unsupported target content
2	Command syntax or option error
3	Capacity or configured resource limit exceeded
4	Host I/O or Typst backend failure
5	Internal invariant failure; request a reproducible report

Exit	Meaning
130	Cancelled by the user

Elm-style diagnostics are structured data

Every warning and error answers four questions: what happened, where, what this means for the result, and what to do next. Prefer a specific edit or an exact flag to advice such as “check your input.” The library returns facts; renderers choose prose layout, color, and machine-readable encoding.

A `Diagnostic` holds stable namespaced code, severity, source ID and byte span, problem kind, bounded typed arguments, consequence, and one or more directions. Directions contain a suggested replacement, a complete argv vector, or a named API field and value. Source text remains borrowed. Fixed inline numeric fields and static message templates allow capacity errors to be reported when report storage is itself full. `Result.primary` is always available by value on failure.

```
UNRESOLVED REFERENCE                                rst.reference.unresolved

  guide.rst:18: See installation_ before continuing.
                ^^^^^^^^^^^^^^

I could not find a target named "installation".
The HTML page has not been published.

Add a target beside the installation section:

.. _installation:

Or change this reference to an existing target name.
```

Rendering calculates line and display column from byte offsets and a bounded line index. A report without source context uses an argv span, resource name, or workspace region. Reports never invent a source position. JSON retains the same code, evidence, consequence, and directions as the text view.

When report storage fills, retain the first failure in `primary`, count suppressed reports, and set an explicit truncation flag. In strict reporting mode, truncation fails the conversion. A bounded excerpt avoids copying a huge source line into the report. The host renderer may use ordinary allocations; a bounded-buffer renderer must remain available for allocation-free embedders.

CLI resource policy

Superseded in part by the draft “Memory budgets and what a build may read”: how a build’s budget is reserved and shared (one budget per build, `--budget`), what it counts and what it names as outside it, and which files a build, a graph, and Typst may read, trusted and under `--untrusted`. The limits below stand.

Initial defaults are proposed, not measured performance claims: 32 MiB per source, 256 MiB total prepared Odin storage, 256 nesting levels, 1,024 stored reports, 1,000,000 AST nodes, and 128 MiB output. Core receives limits explicitly; the CLI selects them. A separate project budget bounds aggregate files and resources. Each exhausted limit names the corresponding flag or API field and a concrete next value when computable. Byte and node capacities are checked independently.

A cancellation probe is checked at stage boundaries and at bounded work intervals. Compressed archives, network retrieval, and office formats are outside this release. Local paths are confined to approved roots with symlink-aware checks. Include and substitution expansion have depth, total-byte, and work limits.

Guidedog: the project builder

NOTE

The draft record “Guidedog: everything Sphinx does, the Odin way” (0006-sphinx-replacement.rst) replaces this section’s project builder: the `guidedog.toml` layout, `init`, and `build --target`. That record is the current statement of the project layout, `conf.toml`, builders, incremental builds, and publication. What this section says about the build cache, publication, confinement, and the themes still holds and is kept for its history; where the two differ, the draft governs.

The full application adds project discovery, document ordering, cross-references, templates, incremental builds, local preview, and publication. It reuses Guidedoc for RST and CommonMark and the Typst adapter for native Typst content.

```
guidedog quickstart docs
guidedog build html docs
guidedog build pdf docs
guidedog serve docs --port 8000
```

These commands are implemented in `lib/project` and `lib/cli/project.odin`; the superseded `guidedog init manual --template atlas` and `guidedog build --target html` forms no longer exist.

Configuration is declarative TOML: source roots, entry document, navigation, targets, language, theme tokens, allowed resources, and build budgets. It is not executable Python. Built-in assets and templates are embedded into the executable. User overrides are explicit files in the project.

Project graph and stable names

Discovery sorts normalized project-relative paths. Parsing collects document IDs, heading targets, named RST targets, citations, includes, and resource dependencies. Resolution constructs a deterministic symbol index and resolves cross-file edges before rendering. Duplicate IDs and cycles are diagnosed with both source sites. URLs are stable project IDs plus anchors, never ephemeral node indices.

For RST and CommonMark, the project host supplies read-only symbol and resource views. Conversion does not open neighboring files. Mixed-source books generate Typst chapter modules for RST and CommonMark and import native Typst chapter modules through a documented content-export convention. An arbitrary standalone `.typ` file can always be built alone; joining it into a book may require adapting page rules and exports. Never promise automatic merging of unrelated Typst programs.

Native Typst chapters declare navigation and exported cross-reference labels in a small project metadata contract. The adapter resolves these against the shared project manifest. It does not guess arbitrary runtime labels from source tokens.

Implemented today: discovery, duplicate-page diagnostics, a cross-file symbol index of labels, terms, domain objects, and documents, and a link pass that resolves references through it: to page URLs for the site, to chapter labels for the book, and to `#<document key>-<anchor>` in `singlehtml`. Native `.typ` documents become site pages through Typst’s HTML export and book chapters through `#include`. The native Typst metadata contract above is not implemented.

A build-cache key includes source and transitive dependency digests, reader version, configuration, renderer version, template, fonts, and Typst version; the implementation also includes the executable’s size and modification time, so a rebuilt `guidedog` never reuses an older conversion. A page’s key also covers what its references resolved to: the link pass records every lookup it makes, and the next build repeats them. Diagnostics are cached with the output they came from and reported again when it is kept. Development caches are host-owned and may allocate. A clean build and a cached build must emit the same artifacts for the same inputs, excluding explicitly requested timestamps.

Responsive HTML and the Typst book

The first HTML theme, proposed name **Atlas**, has a restrained neutral palette, a clear accent color, a readable article measure, and generous spacing. Desktop layout includes project navigation, article, and an on-page outline. On narrow screens, navigation becomes an accessible drawer and the article remains readable at 320 CSS pixels. Wide tables and code scroll within their containers.

Require semantic landmarks, skip links, keyboard navigation, visible focus, proper heading order, image alternatives, high contrast, and reduced-motion support. The page remains readable without JavaScript. Search uses a generated local index; no remote service is required. CSS, fonts, and icons ship with the binary, with license notices. Native Typst HTML is integrated through its supported export and template mechanisms; do not concatenate entire HTML documents into a page shell.

The PDF theme, proposed name **Folio**, is a Typst book template: title page, contents, numbered chapters, running headings, readable body type, restrained code styling, figures, tables, notes, citations, and useful page breaks. Test A4 and Letter, long URLs, wide tables, multilingual text, and embedded fonts. User content is passed as content and data, with a small documented theme-token surface.

Atlas and Folio are implemented and embedded. Desktop and phone-width pages and PDF pages were reviewed as rendered images; a formal accessibility audit remains open. A project `pdf.preamble` file adds package imports and show rules to the book.

Typst and the single executable

Typst’s published compiler library is Rust. The selected design is a small Rust adapter exposing a versioned C ABI, statically linked into the Odin application.

The conversion engine remains Odin. Release builds must prove that the linked binary needs neither an installed `typst` command nor a downloaded package or font. This is an integration decision to validate, not an existing upstream C API.

The ABI accepts pointer-and-length source and resource records, explicit target, limits, and diagnostic callbacks. Rust owns its compiler state and output until an explicit release call; no Rust object layout crosses the ABI. Copying into caller buffers is explicit. Convert panics into failures at the boundary where supported; never unwind through Odin frames. Allocation failure and hard process termination cannot be promised recoverable merely by catching panics.

The adapter implements Typst's resource environment. Project files are read only under the Typst root. **Typst packages are supported**, because publication-quality PDF depends on them and Guidedog writes Typst rather than LaTeX: `@preview` and `@local` imports resolve like the typst CLI, from the package data directory, then the cache, then a Typst Universe download into the cache. A package reads only inside its own directory and has no shell or network access of its own; that is Typst's sandbox, and it makes a package no riskier than a LaTeX package. Versions are pinned by the import itself (`@preview/name:1.2.3`). `--offline` (or `pdf.packages = "offline"`) restricts builds to packages already on disk, for CI and air-gapped machines.

Fonts also follow the typst CLI: extra `--font-path` directories first, then system fonts, then the fonts embedded in the executable as a fallback, so a book can use the fonts installed on the machine. `--embedded-fonts-only` (or `pdf.fonts = "embedded"`) ignores system fonts for byte-reproducible output across machines. Trusted templates are embedded. No other filesystem or network access is delegated to document content. For hard time or memory isolation, a future implementation would invoke an internal worker and apply OS limits. This worker mode is not implemented. Per-platform containment and cancellation must be verified before hostile native Typst is advertised as safely isolated. An in-process compiler is not a sandbox.

The bridge exists in `native/typst_bridge`: ABI version 3, one request carrying root, main file or generated text, target, date, package mode, cache, font mode, and font paths; one owned result released by `gd_typst_free`; panics become an internal-failure status. It builds offline from the cached crates on macOS; Linux vendors OpenSSL statically for the Typst bridge. The complete Unix CLI still links system libcurl. Release builds pass `-define:GUIDEDOG_TYPST_BRIDGE=true`; the default build keeps the development backend.

A development adapter may invoke an installed, pinned Typst CLI with an `argv` array. It is labeled a development backend and does not satisfy the single-binary release gate. Static-link feasibility, binary size, licensing, supported platforms, HTML export fidelity, and worker containment are explicit early technical checkpoints.

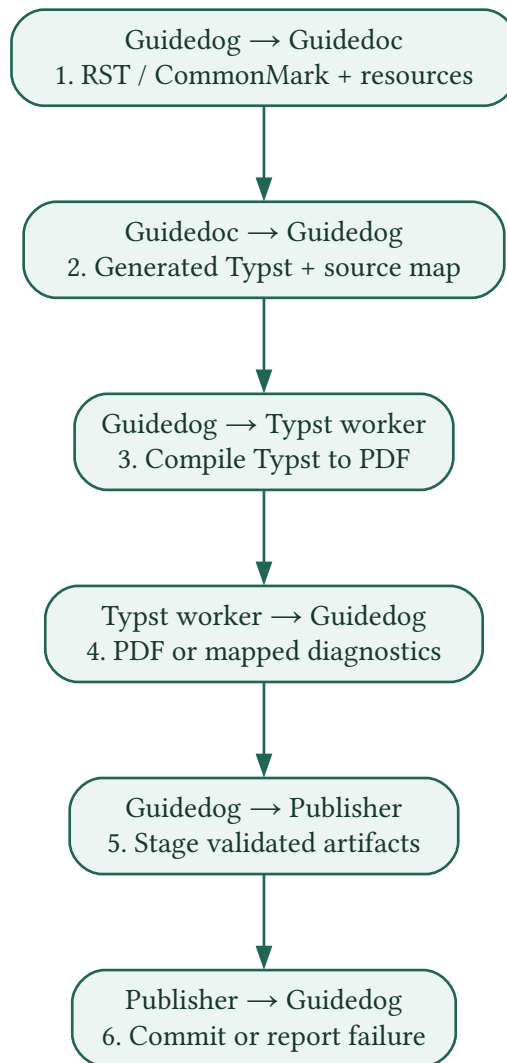


Fig. 4 Native .typ input starts at step 3. Failed compilation never reaches step 5.

Publication is a host transaction

Conversion to a path writes a sibling temporary file, flushes and closes it, then replaces the destination using the platform’s supported rename operation. Refuse existing output unless `--force` is explicit. stdout output cannot be rolled back; finish conversion into staging memory before emitting it.

A website or book with resources is a set of files, so two individual renames do not form an atomic transaction. Stage an entire versioned build directory and switch an application-owned current-version pointer where supported. On platforms without an atomic switch, preserve the previous build and document recovery. An optional provenance manifest records versions, digests, diagnostics, and resources; it belongs to the same staged set. Crash durability requires directory sync behavior where available and is tested separately from logical all-or-nothing publication.

Superseded for project builds by the Publication section of 0006-sphinx-replacement, which builds the whole next output folder beside the live one and switches the two (in one step where the system can), journals the switch so a failure is undone and a crash finished by the next build, publishes the environment with the output, and decides `-w` before publishing. The decision below, whose file-by-file moves could not survive a file that becomes a folder, is kept as history.

Implemented decision. A project build stages each file it writes in a folder beside the output folder, and a file whose bytes are already published is only recorded. When every unit of the build succeeded, the staged files are renamed into place, the files only the previous generation made are removed, and the manifest is saved last. A failed or interrupted build removes the staging folder; the published output is untouched. The switch is file by file rather than one pointer: the output folder stays the path users serve and preview, and unchanged files keep their times. A crash while files are being moved leaves the old manifest, so the next build rewrites exactly the pages that differ.

Delivery phases

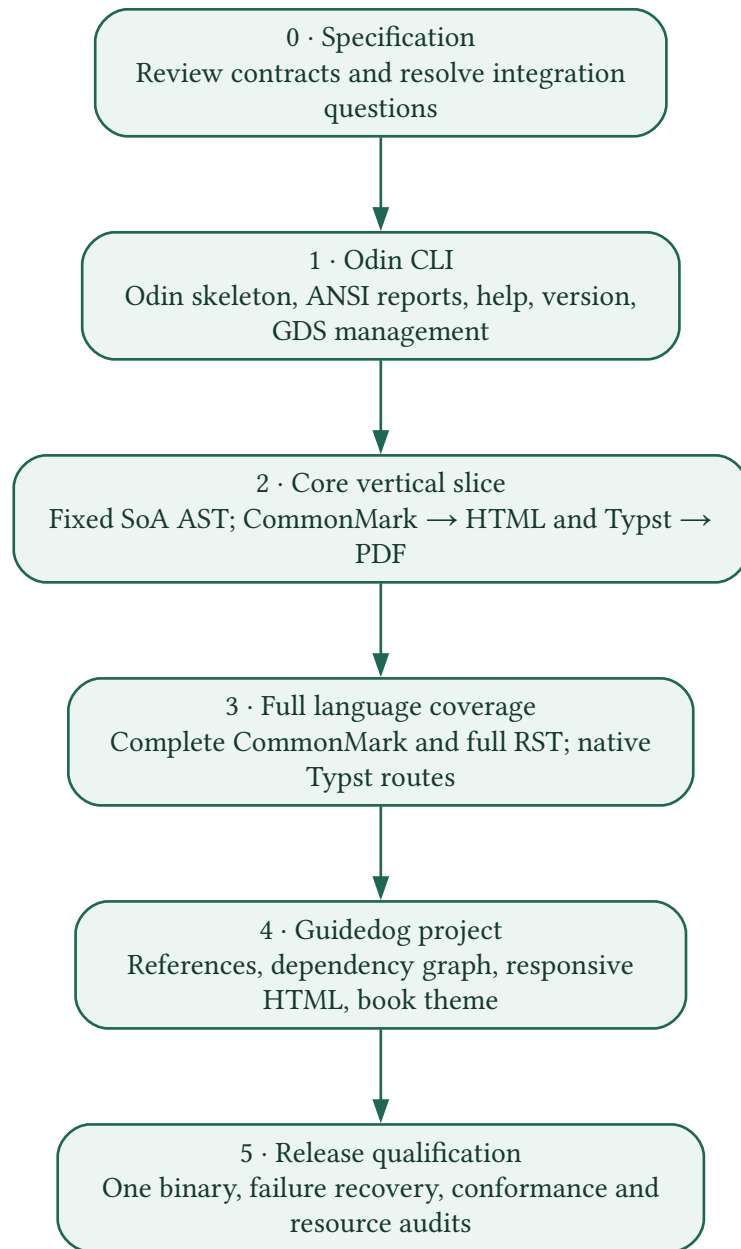


Fig. 5 Each phase proves behavior before adding a larger surface. No phase promises speed.

Phase 2 first uses a development Typst backend while a separate feasibility check validates static linkage. Its CommonMark slice is explicitly partial until the full suite passes. Phase 3 cannot close with a partial RST parser marketed as complete. The release gate requires all declared routes to work from a clean machine without Python or a Typst installation, and without a network download unless a document imports a Typst package that is not cached. Optimization follows working behavior and measured evidence, not an earlier calendar milestone.

Status on 29 September 2026: phases 1 and 2 are complete. Phase 3 is complete for CommonMark and native Typst and partial for RST, as its ledgers record. Phase 4 is implemented except the native Typst chapter contract. Phase 5's single binary works on macOS and Linux; Windows runtime, worker isolation, and font-license review remain.

Security considerations

Source content, terminal excerpts, include paths, and generated Typst are distinct trust boundaries. The rules above require escaping, approved resource roots, explicit raw-content policy, and isolated Typst execution when hard bounds matter. No document may acquire filesystem or network authority by naming a resource. The single-binary packaging requirement is not itself a sandbox guarantee.

Backwards compatibility

The CLI, conversion, and project commands described above are available in this checkout. The diagnostics JSON schema is version 1; changes to it require a new schema version. The project does not emulate every Sphinx extension or configuration script. Full RST support and Sphinx-specific extension support are separate claims. Existing source files must receive useful diagnostics when policy prevents a feature.

Alternatives considered

A separate converter executable would duplicate the future Guidedog command surface. An installed Typst command is useful during development but fails the single-binary release requirement. Reimplementing Typst in Odin would replace a mature compiler with an unrelated language implementation. A narrow static bridge preserves the Odin application and reuses the real compiler. Mandatory sidecars for every byte conversion would improperly move host policy into the core.

Discussion management

`guidedog gds` is part of the first CLI milestone, following docodin's integrated DOD commands and the inherited GDS process. The CLI implements `new`, `promote`, `list`, `show`, `state`, `index`, `check`, `build`, and `recover` according to GDS 0001. GDS HTML/PDF builds use the embedded Typst compiler when it is linked, and otherwise the development backend. Both this repository and documentation projects can select a GDS root with `--dir`.

Acceptance criteria and reference implementation

The CLI must pass help, usage, option precedence, redirection, JSON, ANSI, exit-code, and report-overflow tests. The project builder must pass the engine discussion's conformance and resource tests, the publication failure tests above, and visual review of HTML and PDF templates. Single-binary qualification requires a clean machine with no Python or installed Typst.

Evidence

The reference implementation is `cmd/guidedog` with `lib/cli`, `lib/gds`, `lib/host`, `lib/typst`, `lib/project`, `lib/defaults`, the Guidedoc packages of GDS 0003, and `native/typst_bridge`. On 29 September 2026:

- `tests/` holds 137 end-to-end tests (30 September 2026), among them: discussion workflows, recovery, dates, Typst builds; conversion to HTML and PDF from RST, CommonMark, and Typst; overwrite refusal; JSON and text diagnostics; include confinement; offline package resolution from a private cache; project init, build, incremental rebuild, failed-build preservation, and configuration errors; the preview server; long headings; standard-input includes; exact and one-short capacity for every storage region; memory budgets; publication of whole generations; diagnostics that do not depend on earlier builds; dependencies on tags, nested templates, and moved objects; `singlehtml` addressing; link confinement; and `--untrusted`.
- Package suites (test procedures): core 18, host 5, project 18, i18n 28, gettext 31, jinja 46, highlight 64, graphdog 22, inventory 4, fetch 1, CommonMark 22 (all 652 examples), RST blocks 208, RST inline and resolution 65, Sphinx 73, HTML 28, Typst 24, text 13. Each includes a zero-allocation test where the package is inside the core boundary.
- Every suite passes on macOS arm64 and Linux x86_64, with the installed and the embedded Typst backends. The embedded executable runs with no `typst` on PATH: 64 MB on macOS, 80 MB on Linux. The earlier C-runtime-only claim was incorrect: system libcurl, the C++ runtime, and their transitive dependencies remain. Windows type-checks with and without the bridge.
- Ctrl+C during a PDF build exits 130, leaves no staged file, and publishes nothing.
- `tools/check` passes every source file. A 460 KB, 4,000-section Markdown file converts to HTML in 0.07 s at 33 MB peak memory. Each document and page is converted in its own arena, so a 1,500-document site builds at 15 MB peak. These are observations, not performance claims; every page carries the full navigation, so site size grows with the square of the page count.
- An independent review of the host packages found eleven defects, from anchor sizing to memory growth, output validation, and terminal escaping; all are fixed with tests.

```
odin build cmd/guidedog -out:build/guidedog
cargo build --release --manifest-path native/typst_bridge/Cargo.toml
odin build cmd/guidedog -define:GUIDEDOG_TYPST_BRIDGE=true -out:build/guidedog
odin run tools/check
odin test tests -out:build/guidedog-tests
```

This record is retained as `published`, with incomplete obligations below. GDS 0004 specifies the implemented resource policy; GDS 0006 supersedes the original project surface and publication design. The remaining obligations prevent marking this broader record `committed`.

Decisions still requiring implementation evidence

Hard time and memory isolation for hostile native Typst still needs a worker mode and per-platform verification. The native Typst chapter metadata contract is not implemented; Typst is supported by standalone conversion and PDF templates. Windows runtime validation is still outstanding. Embedded font licenses are recorded with the fonts; they are no longer an open documentation item. Packaging must account for actual runtime dependencies: embedding Typst and static Graphviz does not remove the system `libcurl` link on Unix. A clean machine distribution check remains a separate release gate. The RST ledgers no longer have the previously listed syntax gaps; their pinned tests and conservative reader descriptor state the evidence accurately.

These limits do not prevent a beta for trusted documentation projects on macOS and Linux with the declared dependencies. They do prevent advertising a Windows production release, hostile-input isolation, or a dependency-free Linux executable.

Review history

29 September 2026: separates terminal and host policy from the allocation-free engine; specifies full source-language scope, Typst PDF routes, and later project features. The initial revision preceded the implementation.

29 September 2026, implementation review: record the working GDS CLI, HTML catalog, individual PDFs, journal recovery, local dates, and portable host boundaries. Correct the inherited assignment from 0017 to Guidedog's 0002; zenfmt numbering does not reserve GDS numbers. Record the 18-test evidence and unresolved acceptance gates. Preserve discussion status because the full record is not yet implemented.

29 September 2026, implementation of conversion and projects: record `convert`, `init`, `build`, and `serve`; the embedded Typst bridge; Typst packages and system fonts at the user's direction, replacing offline-only packages and embedded-only fonts; Ctrl+C handling; Linux runtime evidence; and the remaining open items.

29 September 2026, review of the build pipeline: record whole-generation publication with a manifest, recorded lookups and replayed diagnostics, session memory budgets reserved before allocation and shared by reading threads, confinement of linked files, `--untrusted`, and the current test evidence. The cross-file symbol index is implemented.

References

- GDS 0003, Guidedoc: Odin conversion engine.

- GDS 0001, The Guidedog Discussion Process.
- [Odin ANSI package](#).
- [Typst compiler crate 0.15.1](#).
- [Typst HTML export](#).

Lifecycle event

2026-09-29: prediscussion → discussion. Assigned a permanent number and opened for discussion.

Source-format revision — 2026-10-01

RST discussions now use the normal project engine. The design record remains in discussion. See GDS 0001 for the revised authoring contract.

Lifecycle event

2026-10-11: discussion → published. Retain the implemented CLI contract with explicit unfinished worker, native Typst chapter, Windows and packaging obligations.