Guidedog Discussions GDS 0009

Native plugin API and registry

STATE
committed
TYPE
Standards Track
AUTHORS
Vikrant Rathore
CREATED
2026-10-12
UPDATED
2026-10-15
DISCUSSION
Local review; no external discussion URL assigned
LABELS
architecture, library, cli, security, documentation

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Abstract 1
2. Motivation and scope 2
3. Specification 2
4. Rationale and alternatives 7
5. Security and operational considerations 7
6. Backwards compatibility 8
7. Implementation 8
8. Acceptance criteria 9
9. Open questions 10
10. Review history 10
11. References 10

NOTE

GDS 0012, Single executable release, updates the executable boundary for 1.0.0-beta.1. Pages is linked into `guidedog` and needs no registration. References below to `guidedog-pages` describe the earlier process adapter; the process protocol remains available for external plugins.

1. Abstract

Define the plugin foundation before implementing a GitHub Pages plugin. A plugin is ordinary Rust: a struct that implements a trait. Readers and renderers use Guidedoc's existing typed format registry. A new publisher interface consumes a complete, immutable publication. Optional installed publishing executables use a small versioned protocol and a machine registry. A public catalog describes available and planned plugins; catalog membership never installs or executes code.

The API uses direct operations and visible lifetimes. It follows Guidedog's API conventions: `load`, `plan`, `publish`, `status`, and `recover`, with `Drop` for cleanup. It does not copy C naming, introduce classes, require

inheritance, or make a fluent configuration framework. GitHub Pages in GDS 0008 is the first worked publisher and must qualify against this foundation.

2. Motivation and scope

A special command for one hosting provider would leave the next plugin to invent its own API, discovery rules, and credential behavior. A useful foundation should instead tell an author what is borrowed, what is owned, which operation can write, and how a user selects the extension. Its small example should be readable without studying a framework. A normal build must remain predictable when plugins exist.

This record covers existing native format extensions, a proposed publisher API, explicit local registration, and a public plugin catalog. It does not promise Python extensions, a dynamic-library ABI, runtime AST hooks, template execution hooks, or a sandbox. New builder and document-transform extension points require a later design; arbitrary callbacks into the middle of a build would obscure memory, cache, and publication contracts.

GDS 0010 defines that later design for typed source, complete HTML and static-asset hooks. Its development implementation wraps publishers in a general descriptor without changing this record's released publisher contract. It does not add arbitrary AST callbacks or installed custom builders.

Plugins for Guidedog are written in Rust and either linked into a `guidedog` binary or released as native programs. Users need no Python, C, or Rust compiler to run a packaged plugin. Compilers are author/release build tools. Graphviz and the Typst renderer remain native integrations through their bindings and existing platform packaging rules. Source analysis of Python documentation uses the native tree-sitter grammar. Code presentation uses `gd-highlight`. There is no Python plugin loader, interpreter, or Pygments execution in this extension design.

3. Specification

3.1. Three boundaries

Table 20 Extension boundaries

Extension	API and selection	Status
Native reader / renderer	`guidedoc::Reader`, `Renderer`, `Registry`; application supplies the registry	Available since 0.2.3
Linked publisher	`gd_plugins::Publisher` trait; application supplies a `gd_plugins::Registry`	Implemented in 0.6.0
Installed publishing executable	Machine registration and process protocol; user explicitly selects an ID	Implemented in 0.6.0

`guidedoc::Registry` is a value with reader and renderer descriptors. `guidedoc_defaults::registry()` returns the built-in ones. `validate_registry` checks unique IDs within each format kind, versions, and required functions. `find_reader` and `find_renderer` return an `Option`. Applications may compose their own registry and pass it to `gd_project::load` or a conversion session. Do not erase an extension's concrete types into untyped maps.

A document reader is not a publisher. Its memory and resource policy remains the conversion context's policy. Registration grants no credential access and no permission to contact GitHub. Rendering does not acquire publication privileges.

3.2. Direct API rules

Use a concrete struct for each long-lived owner and an options struct for optional settings. Functions take that owner explicitly. `Default` options choose documented defaults, not dangerous behavior. Cleanup is `Drop`, so it runs once, also after a failure. Results carry a `Problem`, not a panic or a process-wide last-error string.

Keep the useful operations few and named for the work they do. Avoid a constructor hierarchy, registration decorators, builder chains, hidden singletons, runtime reflection, or a service locator. A trait is appropriate at a boundary with interchangeable implementations; it is not an excuse to indirect every internal call. The caller owns the lifetime of each session and value.

Follow Rust's naming: `snake_case` functions and `CamelCase` types. A plugin must document every public item's inputs, output lifetime, thread safety, side effects, failure cleanup, and concrete correction hints. Contract comments and an API cheat sheet are required. A list of signatures alone is not an API.

3.3. Publisher interface

The crate is `gd-plugins`. Its public publisher surface is:

Table 21 Publisher surface

Name	Operation	Ownership / effect
`Publisher`	A trait: `info`, `load`, `plan`, `publish`, and the optional `status` and `recover`	The publisher value is its session's state
`Plugin`, `Registry`	`Plugin::publisher(value)`; `Registry::new` validates IDs, API versions and capabilities	An invalid registry cannot exist; no executable starts
`Registry::find`	Find an exact ID	No default guessing or first-match duplicates
`PublisherSession::load`	Validate a selected target and open one publication session	The session owns its settings and state; `Drop` ends it, also after a failed load
`snapshot`	Freeze a successful project's complete publication	Owns private files and manifest; no remote effects
`PublisherSession::plan`	Compare a snapshot with a target and explain changes	The `Plan` owns remote baseline and provider data; reads only
`PublisherSession::publish`	Apply one complete plan and return its verified `Outcome`	Explicit remote writes
`PublisherSession::status`	Inspect the exact operation or generation	Reads only; does not rebuild or retry a write
`PublisherSession::recover`	Reconcile a journal with observed remote state	Explicit recovery; no blind rollback or force reset

A publisher is a struct that implements the trait. `Plugin` holds a prototype and clones it for each session, so sessions never share state. Every hook has typed inputs and a typed result. A single dispatcher over untyped values, and plugin-specific reinterpretation of a host input, are not this interface.

```
pub trait Publisher: Send {
    fn info(&self) -> PublisherInfo;
    fn load(&mut self, options: &LoadOptions) -> Result<Loaded, Problem>;
    fn plan(&mut self, snapshot: &Snapshot, plan: &mut Plan) -> Result<(), Problem>;
    fn publish(&mut self, snapshot: &Snapshot, plan: &Plan) -> Outcome;
    fn status(&mut self, operation: &str) -> Outcome { /* absent: refused */ }
    fn recover(&mut self) -> Outcome { /* absent: refused */ }
}
```

The session wrappers validate inputs, call the selected hook, and enforce the result contract. `load` gives a provider its own typed settings, read from its configuration, not by extending an engine struct or relying on a global variable. Host-made inputs are `#[non_exhaustive]` with constructors, so new fields are not breaking, and new hooks arrive as default trait methods.

One session runs one operation at a time. No operation starts a hidden watcher. Asynchronous use belongs to a caller-owned worker that holds the session, snapshot, and plan until completion and joins before they are dropped. Cancellation is a request to stop; it cannot pretend to undo an accepted remote write.

`State` has `NotStarted` as its default, then `Unchanged`, `Accepted`, `Pending`, `Live`, `Failed`, and `Unknown`. The outcome includes a safe operation ID, generation digest, target URL, and problem. `Accepted` is not `Live`; `Unknown` means the caller must inspect before repeating a write. The CLI maps these states to documented exit outcomes instead of treating every non-error API response as deployment success.

A default outcome is not usable work. Validation requires a complete manifest, generation/target binding, and supported schema before a write. Empty or partly constructed plans cannot mean "delete the site"

or “unchanged”. Success requires a verified terminal state and its required operation identity. This check belongs in the shared session wrappers, not only the Pages command.

3.4. Snapshot contract

A `Snapshot` represents a complete successful generation, not a pathname that might later contain another build. Each publishing invocation builds into a private folder. A publisher never assumes an engine output lock exists. The snapshot owns its private root, manifest, and generation digest. It outlives every plan that borrows it. Dropping a snapshot removes only its own private data after users of that data finish. It does not delete the normal project output.

A manifest entry is a validated relative path, regular-file size, content digest, and explicit mode. Paths and totals are bounded; all arithmetic is checked. No symlinks, hard links, traversal, case collisions, secret stores, build cache, or recovery journal enter a publication. Copying checks file identity and content under the snapshot contract; mutating the frozen copy invalidates it. A failed freeze returns no usable snapshot. Large files stream within the budget. The protocol sends a digest and root owned by the host, not an arbitrary project instruction to read the machine’s filesystem.

3.5. Machine plugin registry

A machine registry records trusted installation choices. It is stored outside projects, with owner-only access, a schema version, and locked atomic replacement of its single file. A registry mutation needs no multi-file recovery journal. Its initial commands are proposed as:

```
guidedog plugin register pages /absolute/path/guidedog-pages
guidedog plugin list
guidedog plugin inspect pages
guidedog plugin run pages -- auth login --profile=github-work --client-id=YOUR_APP_ID
guidedog plugin unregister pages
```

Registration validates ID, absolute executable path, digest, protocol/API version, platform, and declared role before writing an entry. Executing a handshake is already running trusted native code. Registration explains that fact before it runs. A project cannot register an executable, change its path, upgrade its digest, or authorize credential access. A project may name a previously registered publisher ID. Using it still requires an explicit publishing command. Duplicate IDs and changed executables fail with a correction direction, not silent replacement.

Registration accepts native executable packages, not interpreter scripts or shell wrappers. Official catalog admission requires source and reproducible native build provenance. Executable format checks alone cannot prove the source language; that evidence is part of release review. A manifest cannot request installing an interpreter or compiler on the user’s machine.

Registration pins the executable digest, not its pathname alone. Each run checks that pin. Updating an installed binary requires explicit re-registration with a reviewed version. For a bundled executable with sidecar native libraries, the installation manifest pins those files as well. The runtime cannot claim safety against an administrator changing a binary between verification and execution. Platform file-handle execution or immutable installation directories reduce that race. Runtime checks are integrity controls, not a sandbox or trust proof.

`unregister` removes an entry, never the user’s executable, generated site, GitHub repository, or stored OAuth grant. `plugin run` uses an argument vector, not a shell string, and launches only the selected machine entry. It works from any working directory and propagates its exit status. Normal `build`, `convert`, and `serve` never consult the registry to execute a plugin.

A proposed `guidedog publish --publisher=pages --config=pages.toml` drives the shared build/snapshot/plan/publish sequence. `--dry-run` calls only validation, local build/snapshot, and remote reads. It never calls `publish` or `recover`. The provider-specific authentication commands remain behind `plugin run` or the provider’s standalone CLI, rather than making every provider’s login flags core Guidedog flags.

3.6. Process protocol

Installed publishers use a versioned, bounded JSON Lines protocol over dedicated standard input/output. Diagnostic progress goes to standard error. A startup handshake declares protocol version, plugin ID/version, and publisher capabilities. A mismatch fails before a publishing request. No pointer, callback, or native type layout crosses the process boundary.

Each request has a unique ID, operation, target settings, safe operation identity, and snapshot/plan descriptor when needed. A response echoes the ID and contains a validated result or structured problem. Types and enum strings are explicit. A plan's provider data is versioned, bounded JSON, never an executable expression or serialized native pointer. The host owns its stored plan. It signs or hashes the local binding between target, snapshot digest, and baseline, and the plugin checks that binding before writing. Secrets are not protocol fields.

The implementation sets byte limits before reading, maximum nesting, checked number ranges, deadlines, and a cancellation path. Invalid UTF-8, duplicate keys, unknown required operations, unexpected request IDs, truncated replies, and child exit during a write are distinct errors. A failed child after the remote boundary produces `Unknown` with a durable operation ID, not a guessed failure that the host automatically retries. A versioned result schema and fixtures are mandatory. Optional fields can extend protocol 1; incompatible changes require a new version.

The process adapter implements the same publisher semantics as an embedded provider. It is an adapter, not a second publication algorithm. The API version, protocol version, manifest schema, and plugin's release version are separate numbers with tested compatibility rules. Neither source-version equality nor a successful handshake proves that a plugin can safely publish a given plan.

3.7. Upgrades and compatibility

The plugin boundary must survive engine upgrades. A publisher consumes files, manifest entries, target settings, and verified outcomes. It does not receive `guidedoc::Node`, `Workspace`, `gd_project::Project`, template ASTs, cache entries, or compiler-generated layouts. Keep `gd-plugins` independent of the readers and project coordinator. A separate host adapter creates snapshots from project builds. A provider therefore need not import the whole engine merely to publish a snapshot.

There are four distinct contracts: native SDK source API, process protocol, snapshot/plan schemas, and on-disk registry/journal schemas. The Guidedog release version and a plugin's release version identify products, not interoperability. A publisher must not require exact equality with the Guidedog release number. The application and a linked provider are rebuilt together with a qualified Rust compiler. This is source compatibility, not a stable binary ABI; Rust has none. An installed executable remains an independent binary behind the process contract.

Use explicit major/minor pairs for the SDK and protocol. A handshake advertises supported major versions and the minor range for each, plus supported snapshot schemas and named capabilities. The host selects a common major/minor and sends its selection back before any operation. Do not treat an arbitrary higher number as compatible. No overlap gives an explained error before building a remote plan or accessing credentials. The host may build locally without that plugin.

Minor changes add optional fields or operations whose absence preserves prior behavior. Every optional operation has a capability flag, a default trait method, and a documented fallback. Existing required callbacks and their meanings remain fixed. New options have defaults matching old behavior, and host-made inputs are `#[non_exhaustive]` with constructors. Removing an export, adding a required parameter, changing ownership, changing a required outcome's meaning, or requiring a new snapshot field is a major change. A bug fix may strengthen validation of invalid input; it must not silently reinterpret a previously valid plan.

The process envelope has stable request IDs and operation names. Receivers ignore unknown optional fields but reject unsupported required capabilities. Unknown outcome states, invalid numbers, duplicate keys, and an operation the host did not request are errors, never success fallbacks. Error codes and structured fields are compatibility contracts; explanatory prose may improve. A publisher cannot claim `Live` by sending an unfamiliar status that the host coerces to a boolean.

For example, a protocol 1.2 host and a protocol 1.0 plugin may select 1.0 if the requested publication needs only that version's capabilities. If the host needs a new required verification capability, it reports which plugin update is needed. It does not silently weaken verification to keep publication working. A protocol 2-only plugin and a protocol 1-only host cannot publish together. Side-by-side installations let the user keep the compatible version registered while testing an upgrade. Updating Guidedog alone does not rewrite executable integrity pins.

Snapshot schema 1 fixes path normalization, file modes, digest algorithms, and canonical manifest encoding. Hash length-prefixed fields in deterministic path order, not ambiguous concatenated text. Publish encoding test vectors for Unicode paths, empty files, binary content, and deletions. A new digest algorithm can be negotiated only when both sides support it. Engine changes that improve HTML or PDF may change file bytes; they do not change the publication contract. A provider publishes those bytes without trying to parse or regenerate them.

Provider settings and persisted plans have their own versions. A journal records the protocol/schema, provider ID/version, target identity, baseline, candidate, and generation binding required for recovery. An upgrade must either read that journal correctly or refuse it with a concrete instruction to use the previously qualified binary. It must never discard a pending journal as part of migration. Registry migrations write a new file atomically, preserve the original until verified, and do not authorize new capabilities or replace executable pins. A downgrade refuses newer unsupported state without modifying it.

Support and deprecation are published policies. During the beta, label any unstable surface explicitly. Do not advertise API version 1 as stable before its acceptance tests pass. Once stable, keep a major until its documented support window ends. Breaking releases ship a migration guide and overlap where feasible. Security fixes may require blocking an unsafe plugin version. Such a block is explicit, names the reason and replacement, and never silently installs code.

Release CI must exercise current host/current plugin, current host/oldest supported plugin, and previous supported host/current plugin where negotiation promises it. Compile retained native API examples against every supported source contract. Replay protocol fixtures and pending journals across upgrades and downgrades. An engine release may change the AST freely within its own contract without making publication plugins change. A successful current/current run alone is insufficient evidence for an upgrade-compatible interface.

3.8. Public plugin catalog

The repository's `plugins/registry.toml` is a reviewed, versioned catalog. It lists stable plugin IDs, descriptions, maintainers by public names, source URLs, licenses, status, supported API/protocol versions, and verified release artifacts when a plugin ships. It contains no emails or credentials. Planned entries have no installable artifact and are shown as planned, never as released plugins.

The local machine registry and this public catalog serve different purposes. The catalog helps a reader find a plugin. The machine registry records what that reader explicitly chose to run. Refreshing the catalog cannot install, authorize, upgrade, or execute anything. The first implementation may ship a catalog snapshot and support an explicit refresh from a pinned official HTTPS origin. It must not trust a registry URL supplied by document sources.

An eventual installer must verify release checksums, provenance where available, platform, archive paths, complete file manifest, and compatible protocol before registration. Downloaded archives cannot overwrite files outside their installation root or replace the core binary. This proposal initially requires explicit installation followed by registration; automatic package installation is deferred. Third-party entries need review and their own artifact qualification. A listing is not a security certification and does not grant Guidedog's permissions.

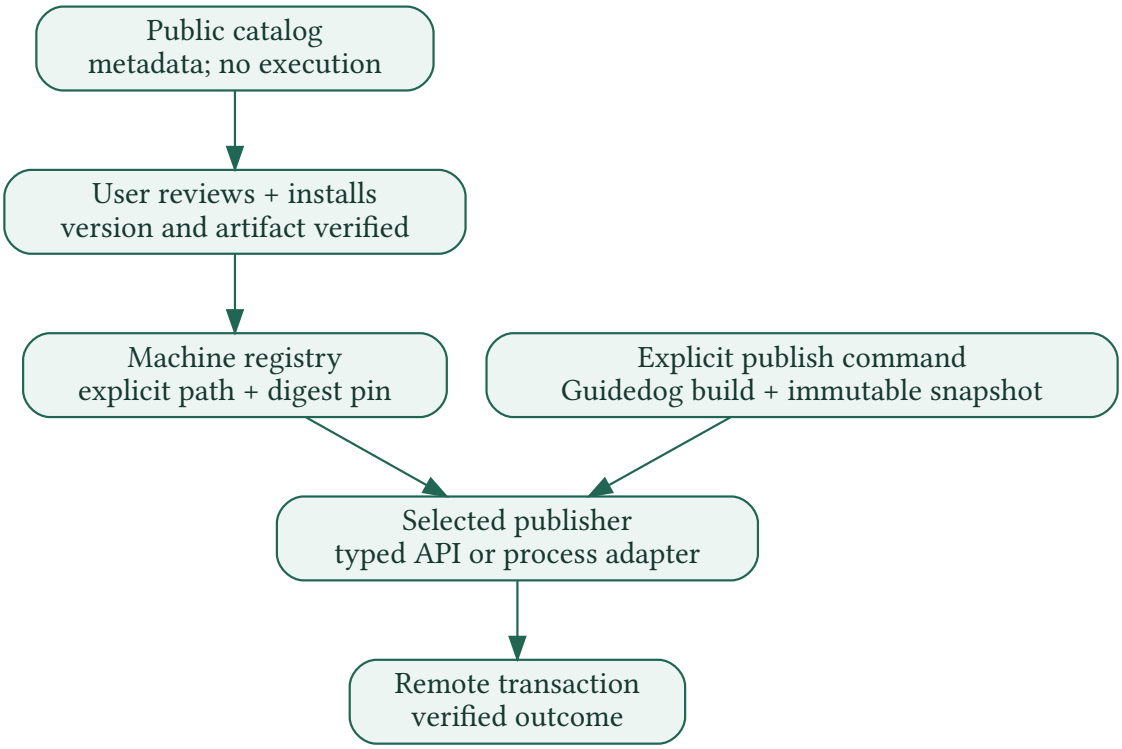

Fig. 11 Discovery describes code; registration chooses code; publication runs it.

3.9. Worked example: GitHub Pages

GDS 0008 implements the `pages` publisher once this interface is qualified. The host builds and freezes. The Pages provider compares the target branch with the snapshot and returns a plan. It creates complete Git objects, moves one ref without force, and confirms the deployment separately. `status` inspects the candidate commit; `recover` reconciles its journal after a lost response. Credentials remain in the provider's platform adapter. A shared credential broker is unnecessary for the first plugin and must not expose tokens to normal readers.

The standalone `guidedog-pages` CLI uses the same library and builds with the installed `guidedog`. It does not link a second document engine. It is not a separate renderer. Registration exposes it through the process adapter. Both routes must produce the same manifest, plan decisions, outcome states, and recovery behavior for fixtures. GitHub Actions publication needs no plugin, machine registry, or personal token.

4. Rationale and alternatives

A trait matches the existing conversion registry's typed descriptors. It gives an embedder a visible list of operations, and the session's lifetime is the value's. Rust has no stable ABI, so a plugin is either linked at build time or run as a program. A process boundary permits optional installation without a dynamic-library ABI. It contains crashes better, but does not limit a native executable's OS permissions. No plugin is loaded implicitly from an untrusted project.

A general event bus or hook-every-step API would make ordering, incremental caches, and resource accounting hard to explain. Specializing the first new surface to complete publication keeps the contract testable. Reusing the existing format registry avoids a second reader/renderer API. A global auto-populated registry would turn a local installation into a change in all builds. Explicit selection preserves reproducibility.

5. Security and operational considerations

Installed native plugins have the permissions of the invoking user. A declaration of capabilities is an interoperability check, not enforcement. The manual must state this when registering code. Project config-

uration may name an opaque credential profile, but only a machine-owned target binding authorizes its use. Catalog metadata and protocol replies cannot create that binding or select another executable or credential. A target edit never silently extends authority. Host auth headers never enter renderer contexts. Provider auth state remains separate from snapshots.
A failure returns no incomplete plan, and cleanup is safe after every failure. Limits bound manifests, JSON lines, process pipes, temporary files, and native dependencies at their real boundaries. There is no promise that a parent budget automatically limits child RSS. Child resource limits are documented and tested by platform before claiming enforcement.

6. Backwards compatibility

`guidedoc::Registry` and `gd_project::load` keep their signatures. No new publisher requirement reaches conversion-only callers. Without explicit publish or plugin registration, Guidedog makes no plugin-authentication or registry network request. A minimal native extension can use current APIs before this proposal is implemented; proposed commands must not appear as available features.

7. Implementation

Guidedog 0.6.0 introduced this record's implementation; `gd-plugins` is the crate, and `docs/manual/internals/plugins.rst` is its design note and wire contract. `crates/gd-plugins/examples/publisher.rs` is a whole publisher, built and run in the tests. The manual teaches the build-plugin example first and supplies a complete GitHub Pages wrapper.

Table 22 Implemented names

Area	Names
Publishers	`Publisher`, `PublisherInfo`, `Capabilities`, `Plugin::publisher`, `Registry::new`, `Registry::find`, `is_valid_id`, `API_VERSION`
Sessions and values	`load_publisher`, `PublisherSession` (`load`, `plan`, `publish`, `status`, `recover`; `Drop` ends it), `Snapshot` (`snapshot`, `verify`; private copies removed on drop), `Plan`, `Outcome`
Outcomes	`State` (`NotStarted`, `Unchanged`, `Accepted`, `Pending`, `Live`, `Failed`, `Unknown`), `Outcome::exit_status`, `EXIT_ACCEPTED` (6), `EXIT_UNKNOWN` (7)
Process protocol	`process_publisher`, `serve_publisher`, `inspect_program`, `Handshake`, `select_version`, `Cancel`, the flag `--guidedog-protocol`
Machine registry	`plugins.toml` in `config_dir()` (`GUIDEDOG_CONFIG_DIR` moves it): `read_machine`, `inspect_entry`, `verify_entry`, `register`, `unregister`
Commands	`guidedog plugin register|list|inspect|run|unregister`; `guidedog publish --publisher=ID --config=FILE [--dry-run] [--wait=SECONDS] [--timeout=SECONDS]`

`Problem` is one pointer wide (a boxed `ProblemData`) and converts to a `gd_report::Report`.
Decisions the implementation made where the draft left room:

- **Linked or served, never loaded.** A publisher is linked into a binary through a `Registry` (as `guidedog` links `gd-pages`) or served as a registered program (`serve_publisher`). There is no dynamic-library or WebAssembly loading.
- **Snapshots** have three modes: `Copy` (a private copy, removed on drop), `Borrow` (checked where it is; the caller keeps the folder unchanged), and `Adopt` (a private folder, removed on drop). The digest is SHA-256 over length-prefixed path, size, and file digest, in byte order of the paths, after the line `guidedog-snapshot 1`. Schema 1 publishes every file as a regular, non-executable file. The child of a process publisher checks the host's snapshot by reading it itself and comparing digests.
- **Plans stay in the session that made them.** A process publisher keeps its plan in the child. The host names it by its ID and generation, and the child refuses another. Plans are not persisted, so a plan cannot outlive its session.
- **One child per session**, answering many requests. `load` starts it and checks its handshake against the registration (ID, version, capabilities, snapshot schema). Dropping the session allows one second for a shutdown reply, closes its input, waits one second, then kills and reaps an unresponsive child. A broken child is stopped directly.

- **The result contract** is enforced in `publish`, `status`, and `recover`: a result without an outcome, an outcome other than `Failed` without its operation, a `publish` result for another generation, or a verified outcome with a problem becomes `Unknown` with `plugins.contract.publish` reads the snapshot again before the publisher writes, and refuses a dry-run session and a plan of another session or snapshot.
- **Registration** reads the program's first bytes (Mach-O, ELF, or PE), pins its SHA-256, runs its handshake once after asking, and refuses an ID registered already. Every `run` and `publish` checks the pin first.

Transport. A writer and a reader thread own the child's pipes. The host waits on channels with one monotonic deadline per exchange and polls a `Cancel` probe, so a blocked write and a drip-fed reply share the same deadline, and no pipe needs to be non-blocking. The handshake is bounded to ten seconds; requests default to five minutes. `--timeout` is separate from the provider's deployment wait. Failure after a possible write is `Unknown`; inspect or recover before retrying. Invalid replies cannot become successful results.
Windows process creation uses an explicit handle list for the three standard streams. Parent pipe ends are not inheritable. A regression test keeps the child running while an unrelated pipe reaches EOF, then requires the child to exit when its own input closes. Ordinary process-wide handle inheritance cannot establish this boundary.
SDK 1.1 adds options and helpers while retaining SDK 1.0 publishers. Process protocol 1.0 and snapshot schema 1 keep their required fields and meanings. Optional snapshot bounds preserve custom accepted limits; an older program can safely refuse a site above its own limits. Source compatibility is not a binary ABI guarantee.
Catalog refresh and installation are outside the shipped interface. The public catalog is reviewed discovery metadata, not an automatic installer. SDK 1.1 remains a beta source interface.

7.1. Command contributions

A plugin of either kind may add commands to `guidedog`'s command line. The GitHub Pages publisher uses it: `guidedog publish pages` and `guidedog pages workflow` are its commands, so `gd-cli` depends only on `gd-plugins`, and `cmd/guidedog` links the plugin. The plugin declares its commands (`commands`: a name of 1 to 4 words, a summary, help and options) and runs them (`run_command` with a `CommandContext`: the host's streams, questions that are never asked without a terminal, a project build by the host, the working folder, the plugin's configuration and cancellation). `guidedog` dispatches an unknown command word, or a `publish` target, to the linked plugin that adds it and then to a registered program, which it starts with `--guidedog-command` and one JSON request on the person's terminal. Handshakes announce commands in two optional fields under a separate command protocol (1.0); the build and publisher protocols and their versions are unchanged. The design and the wire format are in `docs/manual/internals/plugins.rst` ("Command-line extensions", "Command protocol").

8. Acceptance criteria

Table 23 Criteria and evidence

Criterion	Status
A standalone publisher compiles against the SDK; names, comments, cheat sheet, and manual agree.	Done: `crates/gd-plugins/examples/publisher.rs`, `crates/gd-plugins/README.md`, the manual's chapter; every public item has a doc comment.
Reader/renderer and publisher examples compile and run in CI; registry lifetime and malformed registration are tested.	Done.
Failed cleanup, a frozen site that changes, cancellation.	Done for snapshots, sessions, plans, and publications.
Embedded and process publishers pass the same fixtures; handshake mismatches, malformed or truncated lines, and process death never pass for success.	Done for the folder example and for GitHub Pages; a dead child after a write is `Unknown`.
Version negotiation.	Unsupported major versions are refused before load. Current source and process examples pass release qualification.

Criterion	Status
Machine registration: no project-chosen program, no duplicate winner, no silent digest change; atomic, owner-only state.	Done.
Catalog validation and installation.	Discovery metadata ships. Refresh, installation and third-party admission are outside this release; they require their own policy and design.
GDS 0008 uses these APIs as its example.	Done.
The plugin chapter and registry documentation in the maintained languages.	Expanded into a ten-step tutorial in all seven editions for 0.6.1.

The maintainer accepted the implemented scope on 14 October 2026. See Plugin design acceptance, 14 October 2026 for the decision and qualification evidence.

9. Open questions

Establish an official catalog review policy before accepting third-party entries. Dynamic libraries, remote installation, installed builders and mutable doctree transforms remain outside this publisher contract. GDS 0010 defines the separate source, HTML and asset services.

10. Review history

2026-10-12: the maintainer requested a proper plugin API, registry, and manual chapter before the Pages implementation, with raylib-inspired simplicity. This foundation became an explicit dependency of GDS 0008. Promotion opens review; it does not assert that the SDK has shipped.
2026-10-12: the maintainer set 0.6.0 as the release milestone for the completed architecture, with every Guidedog program versioned together.
2026-10-13: implemented in Guidedog 0.6.0, with GDS 0008's GitHub Pages publisher as its first user. The implementation section records the decisions made where this record left room, and what is not yet done.
2026-10-13: the 0.6.1 review corrected ownership across process requests, bounded read and write waits, made registry mutation exclusive even within one process, and added hostile-child regressions. The manual uses the complete tested folder publisher and the real Pages publisher.
2026-10-15: a plugin may add commands to `guidedog`'s command line (see "Command contributions").

11. References

- Guidedoc: the conversion engine, the conversion registry and memory contract.
- Memory budgets and what a build may read, allocation and trust.
- Sphinx-style projects and recoverable publication, complete project publication.
- Publish documentation to GitHub Pages, the first proposed publisher.
- `crates/guidedoc/src/adapter.rs`, typed reader and renderer descriptors.
- `crates/guidedoc-defaults`, the built-in registry.
- `crates/gd-plugins/README.md` and `docs/manual/internals/plugins.rst`, the SDK and its wire contract.
- `plugins/registry.toml`, the proposed public catalog seed.
- `docs/manual/plugins.rst`, the plugin development chapter.
- raylib API reference.

Lifecycle event

```
2026-10-12: prediscussion -> discussion. Open the native plugin API and
registry foundation for review
```

Lifecycle event

```
2026-10-14: discussion -> accepted. Maintainer requested acceptance after
review of the implemented 0.6.2 scope.
```

Lifecycle event

```
2026-10-14: accepted -> committed. Implemented scope reviewed against source
and published qualification.
docs/gds/evidence/plugin-acceptance-20261014.rst; successful 0.6.2 release
run 38434657070.
```