AIT-NATIVE / PRACTICAL PRODUCT GUIDE

User Manual

Version-pinned setup, concepts, practical workflows, concise CLI guides, integrations, operations, and troubleshooting for ait-native 1.1.1.

VERSION	DOCUMENTATION
1.1.1 / 1.1.1	**Revision 4**
CONTENTS	SCOPE
44 guided chapters	**Setup, workflows, integrations, operations**

ait-native.dev/technical/
Published 2026-09-02

Contents

Use the linked page numbers to move through the PDF. Each chapter also links to its exact online route.

START

Technical Documentation

The versioned entry point for ait-native concepts, workflows, commands, integrations, and operations.

AUDIENCE Developers and operators

https://ait-native.dev/technical/v1.1.1/

Use the shortest page that answers the question

This documentation is the technical companion to the basic Docs page. It is fixed to **ait-native 1.1.1** and separates ordinary use, agent-owned workflow commands, integration APIs, and remote operations.

- Start with Getting Started when adding AIT to a repository.
- Read Core Concepts when an AIT record or identifier needs context.
- Open System Architecture and Authority Boundaries to see how local authoring, remote execution, and native embeddings meet the same workflow core without sharing authority.
- Use the complete CLI reference for every public 1.1.1 command, or the shorter workflow-oriented CLI pages for bounded examples.
- Read the Benchmark for the frozen token and elapsed-time result, confidence interval, and protocol boundary.
- Open Remote Infrastructure only when a server and runner solve a concrete coordination need.
- Use the complete Binary DB v0 schema when implementing or auditing fixed records, payloads, indexes, recovery, or conversion.

The operating model

The user initializes the repository once and describes a result. The coding agent reads the generated repository instructions, scopes governed work to an exact sprint item, implements inside the bound workspace, records a Snapshot, and completes the configured closeout path.

```
request and constraints
  -> exact sprint item
  -> Task-bound implementation
  -> repository checks
  -> immutable Snapshot
  -> configured closeout
```

The local workflow does not require a running server. Remote authority and CI are explicit additions.

Version boundary

Every page in this section is tied to the public 1.1.1 source tag. The version panel links to the exact source, Release, distribution contract, and CLI parser used to check this public documentation.

Development-main behavior can be newer than these pages. When a command from a different version disagrees with this documentation, use the documentation for that installed version.

Where to go next

- Feature Workflow
- Regression Workflow
- Benchmark
- Distribution and Release
- Troubleshooting

START

Getting Started

Install 1.1.1, initialize one repository, make a request, and inspect the recorded result.

AUDIENCE Developers

https://ait-native.dev/technical/v1.1.1/getting-started/

1. Install the current release

Choose one verified 1.1.1 route from the Quickstart. The installed native command must report the expected product version before the repository is initialized.

```
ait --version
```

PyPI, npm, Homebrew, apt, and GitHub assets have different package envelopes, but they lead to the same repository workflow. The server remains inactive until it is explicitly configured and started.

2. Initialize one repository

Run initialization from the repository the coding agent will change.

```
cd <your-repository>
ait init
```

AIT creates repository authority and creates or refreshes the effective workflow block in `AGENTS.md`. It does not select a programming language, framework, build command, or test command.

3. Make a normal request

Describe the result and constraints instead of translating the work into AIT bookkeeping.

```
Add an accessible password-visibility control.
Preserve the login API and keyboard behavior, and add the relevant tests.
```

The coding agent follows the repository's generated instructions. For governed work, that includes the sprint item, Task, bound workspace, repository checks, Snapshot, and configured closeout.

4. Inspect the result

Use read-only commands when you need direct evidence.

```
ait status --json
ait diff --stat
ait queue summary
```

An empty workspace after successful closeout is expected. The completed Task and Snapshot retain the recorded result.

Next pages

- Core Concepts
- `ait init`
- `ait status`
- Feature Workflow

START

Core Concepts

Understand the repository objects that connect intent, isolated work, immutable state, evidence, and closeout.

AUDIENCE Developers and integrators

https://ait-native.dev/technical/v1.1.1/concepts/

Repository and Line

A **Repository** is the local AIT authority for one working directory. A **Line** is a named, movable pointer to an admitted Snapshot. `main` is the normal initial Line; a Task receives its own feature Line and bound workspace.

Plan item

A **Plan** records versioned Markdown lineage. In sprint mode, one stable `[plan-ref: ...]` identifies the document and one unchecked checklist item with an exact `[ref: ...]` is the taskable unit.

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

The item states the outcome. It is not an instruction to execute several unrelated changes.

Task, Change, and workspace

A **Task** is the bounded work unit tied to the Plan intent. A **Change** is its reviewable change lineage. The Task-bound workspace separates implementation from the target Line until the result is admitted.

The command that starts the Task prints the exact workspace path. The agent must edit there, not in the canonical repository root.

Snapshot

A **Snapshot** is immutable repository state with authorship and parent history. It records the completed code state; it does not include an unrelated Markdown Plan edit merely to hide documentation lineage inside code history.

Patchset and evidence

In a remote-backed workflow, a **Patchset** is the selected review candidate for one Change. CI, attestation, review, and policy evidence attach to that candidate before final closeout.

Local-only work does not require remote infrastructure. Its configured admission policy still determines what must be verified before completion.

Closeout

Closeout advances the configured target Line, completes the Task, cleans the bound workspace, and applies the repository's sprint-checklist policy. Remote work may require the final Markdown checkbox update and Plan sync as a separate step after code closeout.

Related reference

Continue with Parallel Task Isolation, or open `ait plan`, `ait task`, `ait snapshot`, and `ait workflow` for command guidance.

START

System Architecture and Authority Boundaries

See how external coding clients, native bindings, the ait-native core, Task worktrees, Binary DB, server, and runner connect across local and optional remote authority.

AUDIENCE Developers, integrators, and operators

https://ait-native.dev/technical/v1.1.1/system-architecture/

Read the system as boundaries

ait-native has one native workflow core and two explicit operating paths. The ordinary path is local: an external coding client or embedded host invokes the native interface, implementation happens in a Task-bound worktree, and a Snapshot records the result into local Binary DB authority. Remote infrastructure is optional and extends only recorded state through explicit operations.

Component and authority map

Component boundary	What it owns, and what it does not own
External coding client and Python/Node host	Own the interactive or embedded user experience outside ait-native. They call AIT interfaces but do not replace native workflow or storage semantics.
`ait` CLI, native bindings, and native core	Own command admission, workflow transitions, protocol behavior, and access to recorded authority. Python and Node enter the same core in process rather than implementing a second workflow.
Task-bound worktree and local Binary DB	The worktree is one isolated mutable implementation workspace. Binary DB owns recorded Plan, Task, Change, Snapshot, Line, and related local state. A Snapshot is the boundary between mutable edits and durable AIT history.
`ait-server`	Owns the registered remote Repository authority, shared remote workflow state, typed Worker Jobs, evidence, and offsite preservation of recorded state that reached the server. It does not preserve unsnapshotted local edits.
`ait-runner`	Claims a compatible Worker Job, materializes exact Snapshot content, executes the Repository-declared CI entrypoint in an isolated attempt, and returns bounded evidence. It never becomes Repository authority.

Primary local authoring path

The coding client remains responsible for understanding the request and authoring the implementation. AIT gives that work a stable repository identity and an isolated mutation boundary:

1. The client invokes `ait` through the CLI or a supported native binding.
2. The native core resolves the exact Repository, Plan item, Task, Change, and Task-bound worktree.
3. Code changes remain mutable inside that worktree until a Snapshot is created.
4. The Snapshot and workflow records enter local Binary DB authority; the configured closeout path can then evaluate that recorded state.

Multiple agents remain independent because each active Task owns a distinct worktree and command lock. `AIT_RAM` may accelerate placement, but it does not change the authority boundary: unsnapshotted dirty files are workspace state, while recorded Snapshots remain recoverable AIT state. See Parallel Task Isolation.

Optional remote authority and CI path

Recorded local state reaches `ait-server` only through an explicit configured remote operation. Once received, the server owns the remote Repository copy, shared workflow state, evidence, and Worker Job queue. This enables remote management and offsite recovery of the exact state that reached the server.

For CI, the server commits a typed Worker Job. `ait-runner` claims that job, materializes its referenced Snapshot into isolated attempt storage, runs the Repository-declared entrypoint, and returns bounded results. The server records the resulting evidence; the runner's attempt directory is disposable and is not a second authority.

Remote preservation is not automatic failover and cannot reconstruct dirty or untracked workspace files that were never captured and transferred. See Remote Infrastructure and `ait-runner`: Native CI Execution Plane.

One core through native embeddings

`ait-python` and `ait-node` load the admitted native runtime in process. They provide application-facing entry surfaces without launching a second workflow engine or redefining repository authority. An embedded application must still select the exact Repository context and follow the same Task, Snapshot, remote, and policy boundaries as the CLI.

This diagram is intentionally limited to the current 1.1.1 product boundary. Release-family components, version evidence, and package composition remain defined by Distribution and Release.

START

Benchmark

Read the frozen 100-session-per-treatment AIT versus Git worktree result, confidence interval, acceptance disclosure, and concurrency boundary.

AUDIENCE Developers, evaluators, and technical decision makers

https://ait-native.dev/technical/v1.1.1/benchmark/

Published result

The frozen 1.1.0 campaign compared AIT Task worktrees with Git worktrees over five game-development workloads. It admitted **100 AIT sessions and 100 Git sessions**, with 20 paired sessions per workload and 100/100 accepted in both treatments.

The primary result gives each workload equal weight:

- workload-median token saving: **34.95%**;
- bootstrap 95% confidence interval: **27.85% to 39.77%**; and
- workload-median elapsed-time saving: **21.04%**.

The pooled provider-token totals are a descriptive cross-check rather than the primary estimator: AIT used `46,300,272` tokens versus Git's `70,140,925`, or **33.99% fewer tokens** for AIT in this campaign.

Workload results

Workload	AIT median tokens	Git median tokens	Token saving	95% CI	Pairs
GD-01	213,481.0	317,517.8	32.77%	22.07-41.92%	20/20
GD-02	266,652.6	428,522.2	37.77%	27.59-45.66%	20/20
GD-03	376,821.3	496,364.8	24.08%	12.41-34.32%	20/20
GD-04	580,373.9	915,416.8	36.60%	25.68-45.84%	20/20
GD-05	877,684.7	1,349,224.8	34.95%	22.36-44.82%	20/20

Every workload interval remained above zero in the frozen analysis.

Protocol boundary

Both treatments used the same GPT-5.6 Sol model with maximum reasoning, the same pinned fixtures, the same task prompt and functional acceptance checks, fresh sessions, and provider-reported token accounting. Sessions were run linearly; this campaign did **not** measure concurrent throughput or scaling. It does not guarantee the same saving for every repository, model, or task.

Acceptance disclosure

The campaign executed 201 sessions for 200 admitted observations. The original AIT GD-05 attempt failed the frozen functional check because its change did not preserve `soundEnabled: 0`. The failure remains in the run ledger, was excluded under the frozen acceptance rule, and was replaced once under the same model and fixture pin. The replacement passed and is the admitted observation.

Task-driven concurrency is a separate capability

AIT makes the Task the unit of agent work. Independent Tasks own separate Changes, feature Lines, and worktrees, so commands can run concurrently across many agent sessions. Admission to a shared target Line is revalidated and serialized. This protects the target from stale writes; it does not make concurrent writes to one worktree safe.

See Parallel Task Isolation for the command and authority boundaries. This product capability is not part of the linear benchmark result above.

Frozen evidence

- Human-readable summary
- Machine-readable result
- Session run ledger
- Publication checksums

WORKFLOWS

Feature Workflow

Follow a bounded feature request from sprint item through implementation, verification, and closeout.

AUDIENCE Developers and coding agents

https://ait-native.dev/technical/v1.1.1/workflows/feature/

Scenario

The feature example matches the interactive Demo: add an accessible password-visibility control while preserving the login API and keyboard behavior.

The user supplies the request. The coding agent carries the repository workflow.

Scope the work

The agent writes one sprint card with one exact taskable item.

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

It then starts the bound Task from that exact item.

```
ait task start \
  --from docs/sprints/password_visibility.md#login/password-visibility/implement \
  --intent "Add accessible password visibility control"
```

The output supplies the Task, Change, and workspace path.

Implement in the bound workspace

The agent enters the printed path, updates the control and focused tests, and runs the repository-owned validation. AIT does not guess a framework or invent a test command.

```
src/LoginForm.tsx
tests/LoginForm.test.tsx
3 tests passed
```

Record the result

After the checks pass, the agent records the code state.

```
ait snapshot create --message "Add accessible password visibility control"
```

Remote-backed work then uses the guided readiness command until the selected Patchset has the required CI, attestation, review, and policy evidence, and uses the reviewer-owned finish surface for atomic remote closeout.

```
ait workflow ready <change-id> --apply
ait workflow finish <change-id> --apply
```

Complete and verify

The agent finishes with the Task command from the generated repository instructions, then verifies Task, workspace, Line, and sprint-item closeout.

```
ait task finish <task-or-change-id>
ait task audit <task-id>
```

Because the example already created its final Snapshot, `task finish` reuses the clean Line head. When local work is still dirty, use `ait task finish <task-or-change-id> --message <message>` so the same operation creates the final Snapshot before closeout.

Use the Interactive Demo to watch the same flow, or open the `ait task` reference for command details.

WORKFLOWS

Regression Workflow

Identify the responsible revision, isolate the repair, add a regression test, and close the new Task.

AUDIENCE Developers and coding agents

https://ait-native.dev/technical/v1.1.1/workflows/regression/

Scenario

The existing password control no longer responds correctly to keyboard activation. The repair must preserve the intended behavior and add a test that fails on the responsible revision.

Identify the responsible state

Before choosing the repair, inspect the relevant line or bounded range.

```
ait blame src/LoginForm.tsx --line 84
```

The result connects the source line to its responsible Snapshot and available Plan lineage. This evidence answers where the behavior entered history; it does not automatically decide the repair.

Create a new repair Task

Record a new sprint item for the regression instead of reopening completed work.

```
# Password Keyboard Regression [plan-ref: login/password-keyboard-regression/root]

- [ ] Restore native keyboard activation and lock it with a regression test. [ref:
  login/password-keyboard-regression/repair]
```

```
ait task start \
  --from docs/sprints/password_keyboard_regression.md#login/password-keyboard-regression/repair
    \
  --intent "Restore password-control keyboard activation"
```

Prove the repair

The agent changes only the bounded behavior, adds the regression test, and runs the repository checks in the Task workspace.

```
4 tests passed
```

It records a new Snapshot and follows the same configured readiness and closeout path as any other governed change.

```
ait snapshot create --message "Restore password-control keyboard activation"
ait task finish <task-or-change-id>
```

Recovery rule

If blame evidence is incomplete or the repair grows beyond the stated scope, stop and re-scope the Task. Do not overwrite unrelated workspace changes or reinterpret a prior completed Task as active work.

See `ait blame` and Troubleshooting.

CLI · USER AND INSPECTION

ait init

Create or reinitialize AIT repository authority and effective agent instructions.

AUDIENCE Users

https://ait-native.dev/technical/v1.1.1/cli/init/

Role

Normally run by the **user** once from the repository that a coding agent will change.

Purpose

`ait init` creates missing local repository authority and creates or refreshes the effective AIT workflow block in `AGENTS.md`. Reinitialization preserves valid existing authority instead of replacing it.

Syntax

```
ait init
ait init --json
```

Important optional inputs include the initial repository name, default Line, policy profile, author mode, and a bounded repair flag for incomplete existing structure. Ordinary first use needs no option.

Example

```
cd ~/src/example-app
ait init
```

After success, open `AGENTS.md` or run `ait config show` to inspect the effective workflow routing. Initialization does not start `ait-server` and does not infer a programming language or validation command.

Output

Text output summarizes the created or preserved repository and guidance. JSON is the stable machine-readable result for automation.

Failure and recovery

If the directory contains malformed or partial AIT state, keep the error and inspect the existing `.ait` structure before using a repair option. Never replace authority merely to make initialization succeed.

Related

- Getting Started
- `ait config`
- `ait status`

CLI · USER AND INSPECTION

ait config

Inspect effective workflow routing and apply explicit repository configuration changes.

AUDIENCE Users and operators

https://ait-native.dev/technical/v1.1.1/cli/config/

Role

Normally read by the **user or coding agent**. Configuration changes should be made only when the repository owner has selected the new behavior.

Purpose

`ait config` reports effective repository routing and applies explicit workflow presets or bounded overrides. A successful relevant change also refreshes the generated AIT block in `AGENTS.md`.

The command surface is intentionally narrower than the persisted file schema. For every supported root field, nested field, owner, absence rule, and worktree overlay, use Appendix: `.ait/config.json`.

Syntax

```
ait config show
ait config show --json
ait config set --workflow-mode solo_local
ait config set --sprint on
```

For a remote-backed individual workflow, the corresponding primary preset is `solo_remote`. Prefer a primary preset before adding a narrow override.

Example

```
ait config show --json
ait config set --workflow-mode solo_local
ait config show
```

Inspect the generated guidance after the write. The printed task, Plan, and closeout commands are the repository-specific route.

Output

Text output shows the effective values most useful for a decision. JSON provides the complete machine-readable configuration projection.

Failure and recovery

Changing a default does not move an existing Task, Change, Plan binding, or workspace to the new scope. Complete or recover existing lineage under its recorded contract before assuming the new default applies.

Related

- Appendix: `.ait/config.json`
- `ait init`
- `ait workflow`
- Remote Infrastructure

CLI · USER AND INSPECTION

ait status

Inspect the current Line, repository state, workspace state, remotes, and worktree health.

AUDIENCE Users and coding agents

https://ait-native.dev/technical/v1.1.1/cli/status/

Role

Safe for the **user or coding agent** as the first repository inspection.

Purpose

`ait status` summarizes the current Line, head Snapshot, workspace state, configured remotes, repository identity, and worktree health. It is read-only with respect to product lineage.

Syntax

```
ait status
ait status --json
```

Example

```
ait status --json
```

Use JSON when a coding agent must decide whether the canonical root or a Task-bound workspace is active. Use the compact text view for a quick human orientation.

Output

Typical fields include the current Line, default remote, head Snapshot, workspace changed counts, repository name, and worktree hygiene. Operational health detail can change without creating a new Snapshot.

Failure and recovery

If repository discovery fails, confirm the intended repository root and run `ait init` only when no valid authority exists. If status reports a dirty workspace, inspect `ait diff` before starting or recovering a Task.

Related

- `ait diff`
- `ait queue`
- Troubleshooting

CLI · USER AND INSPECTION

ait diff

Compare the current workspace with the current Line head using bounded path filters.

AUDIENCE Users and coding agents

https://ait-native.dev/technical/v1.1.1/cli/diff/

Role

Safe for the **user or coding agent** before a Snapshot, repair, or recovery decision.

Purpose

`ait diff` compares the current workspace with the current Line head. It does not select arbitrary Snapshot ranges; use `ait snapshot diff` for immutable Snapshot comparison.

Syntax

```
ait diff
ait diff --stat
ait diff --name-only
ait diff [<path>...]
```

Path inputs are exact lexical filters, not shell-style patterns. `--stat` and `--name-only` are alternate summaries.

Example

```
ait diff src/LoginForm.tsx
```

The result classifies modified, missing, and untracked paths and bounds large or binary content rather than printing an unlimited payload.

Output

The normal text view emphasizes changed paths and useful content. `--json` provides the machine boundary when automation needs exact classifications. Sprint Markdown is tracked through Plan lineage and does not appear as code workspace content.

Failure and recovery

If an expected file is absent, inspect the Task workspace and Line before restoring anything. Do not overwrite unrelated changes to force a clean diff.

Related

- `ait status`
- `ait snapshot`
- Troubleshooting

CLI · USER AND INSPECTION

ait blame

Trace a file or line range to the responsible Snapshot and Plan revision before a repair.

AUDIENCE Users and coding agents

https://ait-native.dev/technical/v1.1.1/cli/blame/

Role

Run by the **coding agent or user** before choosing a repair for an identified regression.

Purpose

`ait blame` attributes a path or bounded line range to the responsible Snapshot and available Plan revision lineage. It connects code state with the recorded change history without mutating the workspace.

Syntax

```
ait blame <path>
ait blame <path> --line <number>
ait blame <path> --start <line> --end <line>
```

Example

```
ait blame src/LoginForm.tsx --line 84
```

Use the smallest range that answers the question. A whole-file query is useful when the relevant region is not yet known.

Output

The result identifies the responsible Snapshot and bounded attribution evidence. Plan information is included when that source state is connected to versioned Markdown lineage.

Failure and recovery

Missing attribution is evidence, not permission to guess. Confirm the path, Line, and imported history; then record the repair as a new bounded Task.

Related

- Regression Workflow
- `ait snapshot`
- `ait task`

CLI · USER AND INSPECTION

ait doctor

Discover focused diagnostic surfaces without changing repository authority.

AUDIENCE Users and operators

https://ait-native.dev/technical/v1.1.1/cli/doctor/

Role

Used by the **user or operator** when normal status evidence identifies a storage, runtime, Plan-authority, or remote dependency problem.

Purpose

`ait doctor` is a namespace of focused diagnostics. The available diagnostic topics are versioned, so begin with the installed command's help instead of copying an unrelated maintenance command.

Syntax

```
ait doctor --help
```

Worktree-specific diagnosis is available under `ait worktree doctor`; it is separate from the top-level doctor topics.

Example

```
ait status --json
ait doctor --help
ait worktree doctor --refresh --json
```

Select only the diagnostic that matches the status finding. Diagnostics should not become a speculative repair sequence.

Output

Doctor output reports the inspected boundary, evidence, and any bounded next action. Preserve it when escalation is required.

Failure and recovery

Do not run a destructive cleanup merely because a doctor reports stale or partial state. Resolve the exact owner, identifier, and path first, then use the recovery command supplied for that condition.

Related

- `ait status`
- Troubleshooting

CLI · AGENT WORKFLOW

ait plan

Record versioned Markdown lineage and synchronize exact sprint items in the configured scope.

AUDIENCE Coding agents and maintainers

https://ait-native.dev/technical/v1.1.1/cli/plan/

Role

Normally run by the **coding agent** under the exact scope printed in the repository's generated instructions.

Purpose

`ait plan` records versioned Markdown lineage. In sprint mode, one exact open checklist item becomes the Task binding; ordinary documentation can be synced without becoming a Task.

Syntax

```
ait plan sync <markdown-file-or-dir> --local
ait plan sync <markdown-file-or-dir> --remote origin
```

Use only the scope that matches the effective repository configuration. The initial `ait task start --from` invocation owns exact-file synchronization for its bound sprint item.

Example

```
# Password Visibility [plan-ref: login/password-visibility/root]

- [ ] Add the accessible control and focused tests. [ref: login/password-visibility/implement]
```

```
ait plan sync docs/architecture.md --local
```

Output

Sync reports the Plan identity, revision, action, and publication scope. Plan views can inspect items and revisions without changing them.

Failure and recovery

Do not copy a Plan ID into `task start` or hide a Markdown checkbox update in a code Snapshot. When remote closeout leaves the checklist update separate, edit the exact bound item and sync that card in the configured remote scope.

Related

- Core Concepts
- `ait task`
- Feature Workflow

CLI · AGENT WORKFLOW

ait task

Start Plan-bound work, inspect readiness, and complete the Task through its configured scope.

AUDIENCE Coding agents and maintainers

https://ait-native.dev/technical/v1.1.1/cli/task/

Role

Normally run by the **coding agent**. The user states the desired result and constraints rather than manually driving Task bookkeeping.

Purpose

`ait task` binds sprint intent, Change lineage, a feature Line, a managed workspace, and final closeout. In sprint mode, start from one exact taskable Markdown item.

Syntax

```
ait task start --from <sprint-card>#<exact-ref> --intent <intent>
ait task audit <task-id>
ait task finish <task-or-change-id>
ait task finish <task-or-change-id> --message <message>
```

Example

```
ait task start \
  --from docs/sprints/password_visibility.md#login/password-visibility/implement \
  --intent "Add accessible password visibility control"
```

The final output includes the Task, Change, and exact `cd` command. All code work belongs in that printed workspace.

Audit and closeout

`ait task audit` reads the Task's readiness and closeout state. `ait task finish` uses the recorded workflow scope. Dirty local work requires `--message` and is captured as the final Snapshot by that operation; clean local work omits the option and reuses the current Line head. A remote Task rejects `--message` and must already have a selected Patchset with completed readiness evidence.

The public closeout surface uses `ait task finish`, while reviewer-owned closeout uses `ait workflow finish`. `Land` remains only the name of the internal atomic operation and its durable records; it is not a public command.

Failure and recovery

If startup partially succeeds, keep the printed identifiers and inspect the Task before retrying. If a workspace is missing, use the worktree recovery surface; do not create a second Task for the same open sprint item unless the original lineage is explicitly resolved.

Related

- Parallel Task Isolation
- `ait plan`
- `ait snapshot`
- `ait workflow`

CLI · AGENT WORKFLOW

ait snapshot

Record immutable repository state and inspect Snapshot content or ancestry.

AUDIENCE Coding agents and maintainers

https://ait-native.dev/technical/v1.1.1/cli/snapshot/

Role

Normally created by the **coding agent** after the bounded implementation and repository checks are complete. Snapshot inspection is safe for users.

Purpose

`ait snapshot` records immutable repository state and exposes content, comparison, ancestry, and recovery queries. A Snapshot keeps its authoring Line identity even when a later Change or closeout target moves.

Syntax

```
ait snapshot create --message <message>
ait snapshot show <snapshot-id> --files
ait snapshot ancestry <snapshot-id> --ancestors
```

Snapshot comparison uses two immutable identifiers:

```
ait snapshot diff <old-snapshot-id> <new-snapshot-id>
```

Example

```
ait snapshot create --message "Add accessible password visibility control"
```

Run this from the Task workspace after validation. A normal Snapshot records code state; separately authored sprint Markdown remains Plan lineage.

Output

Create returns the new Snapshot and parent. Show can include the full file inventory, while ancestry queries are bounded unless complete output is explicitly requested.

Failure and recovery

A dirty or misplaced workspace should be inspected with `ait status` and `ait diff` before retrying. Revert and replay operations are separate explicit surfaces; preview their effect before replacing workspace content.

Related

- `ait diff`
- `ait task`
- Core Concepts

CLI · AGENT WORKFLOW

ait queue

Read a compact inventory of active work, reviews, remotes, and worktree health.

AUDIENCE Users and coding agents

https://ait-native.dev/technical/v1.1.1/cli/queue/

Role

Safe for the **user or coding agent** when orienting active local and remote work.

Purpose

`ait queue` is a read-only inventory composed from Tasks, Changes, workspace, worktrees, reviews, and configured remote state. It does not own a separate lifecycle queue.

Syntax

```
ait queue summary
ait queue summary --remote origin --json
```

Example

```
ait queue summary
```

The compact text result reports shared Tasks, ready candidates, review inbox, local drafts, workspace changes, and worktree hygiene.

Output

Use `--remote <name>` to include the selected remote authority and `--json` for the complete machine-readable projection. Any ready indication is orientation evidence; use `ait task audit` or the guided workflow surface before mutation.

Failure and recovery

A remote read error can appear beside otherwise valid local inventory. Keep the error visible, check the configured remote and server health, and avoid treating missing remote rows as completed work.

Related

- `ait status`
- `ait task`
- `ait workflow`

CLI · AGENT WORKFLOW

ait workflow

Classify bounded edits, prepare remote readiness evidence, and inspect closeout state.

AUDIENCE Coding agents and operators

https://ait-native.dev/technical/v1.1.1/cli/workflow/

Role

Normally run by the **coding agent**. Operators may use the same decision surfaces to inspect a remote gate without bypassing it.

Purpose

`ait workflow` classifies bounded edits, prepares remote Patchset readiness, and inspects or resumes closeout. Effective scope and exact commands come from the generated repository instructions.

Syntax

```
ait workflow guide [topic]
ait workflow reconcile --dry-run --json
ait workflow ready <change-id> --apply
ait workflow finish <change-id> --apply
```

`workflow ready` and `workflow finish` are text-only decision surfaces. Do not append `--json` to them.

Guide and reconciliation

`workflow guide` prints one supported playbook. `workflow reconcile` inventories cross-object Task, Change, Line, worktree, Land, and Plan-binding state. Its default and explicit `--dry-run` forms do not mutate Plan state; use `--apply` only for a reviewed reconciliation action.

Remote readiness

`workflow ready --apply` can create or synchronize the selected Patchset and advance required CI, attestation, review, and policy state. It stops with one exact next action when a gate needs separate work.

The explicit remote reviewer-owned closeout is:

```
ait workflow finish <change-id> --apply
```

It checks the selected Patchset's review and policy requirements before the safe atomic finish. Configured local closeout, or remote closeout after all readiness evidence is already complete, can instead use:

```
ait task finish <task-or-change-id>
```

Failure and recovery

Do not substitute local checks for required remote CI, append unsupported output flags, or switch authority to avoid a blocker. Re-run the decision surface after the owning evidence changes and preserve the same Change ID.

Related

- `ait task`
- `ait queue`
- Remote Infrastructure

INFRASTRUCTURE AND INTEGRATIONS

Parallel Task Isolation

Run multiple coding-agent Tasks concurrently through isolated worktrees, AIT_RAM fan-out, immutable evidence, and atomic Land.

AUDIENCE Developers, coding agents, and operators

https://ait-native.dev/technical/v1.1.1/workflows/parallel-task-isolation/

The operating model

AIT permits **parallel authoring with serialized admission**. Multiple coding agents may work at the same time, but they do not share one mutable checkout and they do not write directly to `main`.

Each governed request receives its own Task, Change, feature Line, and bound worktree. The agent records immutable Snapshots in that lineage. A candidate is admitted to the target Line only after its exact Patchset has the required evidence and the target head has been checked again.

This division is important:

- Task-bound worktrees and command locks prevent workspace operations from colliding.
- immutable Snapshots prevent a reviewed revision from changing underneath its evidence;
- rebase and target-head checks prevent a stale candidate from overwriting newer work; and
- atomic Land prevents a last-writer-wins update to the shared target Line.

AIT does not make it safe for two agents to edit the same worktree. High parallelism comes from creating more isolated Task worktrees, not from sharing one directory more aggressively.

High-concurrency command fan-out

Independent agents may issue `ait task start`, inspection, Snapshot, test, and readiness commands at the same time for different Tasks. Each command resolves its Task-owned worktree and lineage, so a large queue can fan out without one global mutable checkout. Repository command locks protect AIT's authority updates; they do not turn unsynchronized writes to ordinary files in one worktree into a safe operation.

Finishes that target the same Line are deliberately not a throughput race. Each candidate revalidates the current target head, rebases or stops when needed, and advances the target atomically. This is the boundary behind the product's high-concurrency Task support: author and validate independently, then serialize admission to shared authority.

One Task owns one mutable checkout

Object	Parallel-execution responsibility
Sprint item	Gives one Task a distinct requested outcome and exact `[ref]`.
Task	Owns the bounded unit of work and closeout state.
Change	Owns the mutable review lineage for that Task.
Feature Line	Keeps the Task head separate from the target Line.
Bound worktree	Gives one agent a physically separate writable checkout.
Snapshot	Freezes one immutable revision of that checkout.

Object	Parallel-execution responsibility
Patchset	Selects the exact Snapshot evaluated by CI, review, attestation, and policy.
Land	Revalidates and atomically advances the target Line.

`ait task start` creates and binds these working identities. Its output prints the exact `cd` command. The agent must enter that path before modifying code.

```
ait task start \
  --from docs/sprints/login.md#login/password-visibility \
  --intent "Add accessible password visibility"
```

A second agent starts a different open item and receives a different Task, Change, Line, and worktree.

```
ait task start \
  --from docs/sprints/login.md#login/session-regression \
  --intent "Repair session renewal regression"
```

Do not assign the same sprint item to two agents. 1.1.1 validates that an item is open and taskable before Task creation, but it does not claim that one Plan item has a complete cross-client, server-owned uniqueness lock. Parallel Tasks should begin from distinct exact item references.

Two agents, one target Line

Assume both Tasks fork from `main` at Snapshot `S0`:

```
main @ S0
|
+-- Task A -> Change A -> feature Line A -> worktree A
|               `-> Snapshot A -> Patchset A
|
`-- Task B -> Change B -> feature Line B -> worktree B
                `-> Snapshot B -> Patchset B
```

The agents implement and test concurrently inside their printed paths.

```
# Agent A, inside worktree A
ait snapshot create --message "Add password visibility control"
ait workflow ready <change-a> --apply

# Agent B, inside worktree B
ait snapshot create --message "Repair session renewal regression"
ait workflow ready <change-b> --apply
```

Suppose Task A finishes first and moves `main` from `S0` to `S1`. Task B does not overwrite `S1` with a candidate based on `S0`.

```
Task A: main S0 -> S1

Task B readiness:
  base S0 != current main S1
  |
  +-- clean rebase -> refresh Snapshot/Patchset evidence -> eligible Land
  |
  `-- conflict -> stop with main unchanged -> explicit resolution required
```

For a clean bound worktree, guided readiness can rebase it onto the current target before publishing the next candidate. The resulting candidate is new content and must carry evidence for that exact selected Patchset; an earlier CI pass is not reused as proof for the rebased revision.

If rebase conflicts, AIT records the paused state and leaves the target Line unchanged. Inspect and either continue or abort explicitly:

```
ait worktree status --json
ait worktree rebase --continue
# or
ait worktree rebase --abort
```

After conflict resolution, record and prepare the resulting revision as directed by the current workflow output. `ait task finish` consumes an already ready selected Patchset. If the target head changes again before admission, the operation fails closed instead of hiding the intervening result.

Why `AIT_RAM` is part of the strategy

Filesystem isolation is correct but can be expensive when every Task restores an entire repository onto disk. The `AIT_RAM` strategy places managed Task worktrees on a verified memory-backed filesystem and maintains one reusable read-only `main-seed` for rapid fan-out.

`AIT_RAM` is an execution and cache layer. It is **not** the AIT authority, a backup, or a replacement for `ait-server` recovery. Task, Line, Snapshot, and other repository authority remains in the persistent repository state or the configured server. Only the materialized checkout is placed in volatile memory.

```
Persistent repository
|
+-- .ait/                          Task, Line, Snapshot authority
+-- .ait-worktree-links/task-a --+
`-- .ait-worktree-links/task-b --+---- stable local aliases
                                 |
/Volumes/AIT_RAM/                |    memory-backed execution
`-- .ait-repos/<repository-key>/ |
    |                            |
    +-- main-seed                |    read-only main Snapshot
    +-- task-a <-----------------+    writable Agent A checkout
    `-- task-b <-----------------+    writable Agent B checkout
```

The Repository key is derived from the authoritative repository path. It partitions the shared memory root so different repositories do not collide even when their Task worktree names are similar. The default stable alias root is `.ait-worktree-links` inside the repository.

Inspect the resolved paths rather than assuming them:

```
ait config show --json
ait worktree list --json
```

The configuration projection reports `task_worktree.memory_root`, the derived `ephemeral_root`, the alias root, and any main-seed RAM budget.

Effective 1.1.1 memory defaults

The defaults are deliberately literal; 1.1.1 does not derive them from a percentage of host memory.

Control	Built-in value	Effect
macOS managed RAM volume	8 GiB (`8,589,934,592` bytes, or `16,777,216` 512-byte sectors)	Capacity of the built-in `/Volumes/AIT_RAM` image.
Linux or Windows RAM capacity	No AIT allocation default	AIT uses the capacity of an already-mounted `tmpfs`/`ramfs` or `DRIVE_RAMDISK`.
Repository `main_seed_ram_max_bytes`	Unset (`null`)	No Snapshot-size cutoff is applied to RAM eligibility. This does not make physical RAM unlimited.
Task-placement free-space threshold	None	1.1.1 has no byte or percentage threshold that automatically moves a new Task to SSD.
`ait doctor memory-root`	Read-only	The diagnostic never mounts, provisions, repairs, or creates a root.

Use `ait config show --json` and `ait doctor memory-root --json` to distinguish stored Repository policy from the host's current capacity. The former reports the typed memory-root selection and repo-local seed limit; the latter reports requested capacity, total bytes, available bytes, and the built-in zero-byte validation floor with each value's source.

Memory-root validation and provisioning

Before relying on RAM placement, prove that the configured root is genuinely memory-backed and writable:

```
ait doctor memory-root
```

The diagnostic is always read-only. Managed Task placement is a separate path: while resolving a **new** worktree on macOS, it may provision the configured or built-in RAM volume. If that attempt cannot produce a usable root, placement continues to the persistent-disk fallback. A path merely named `AIT_RAM` is not accepted as proof that the storage is memory-backed.

Platform	Required proof	Provisioning behavior
macOS	Exact `/Volumes/<name>` mount backed by a writable `hdiutil` RAM image with the expected APFS volume identity.	New-Task placement can provision the typed or built-in volume; the diagnostic never does.
Linux	An existing mount whose filesystem type is `tmpfs` or `ramfs`.	AIT validates but does not create the system mount.
Windows	An existing root reported as `DRIVE_RAMDISK`.	AIT validates but does not create the RAM disk.

The repository configuration uses these typed fields:

Field	Purpose
`task_worktree.memory_root.kind`	Platform proof: `macos_ram_volume`, `linux_memory_root`, or `windows_ramdisk`.
`task_worktree.memory_root.root`	Exact absolute memory-root path.
`task_worktree.memory_root.volume_name`	macOS RAM-volume label; not used by Linux or Windows.
`task_worktree.memory_root.sector_count`	Positive macOS `ram://` sector count. Omitted uses `16,777,216` sectors.
`task_worktree.main_seed_ram_max_bytes`	Optional nonnegative default-Line Snapshot-size ceiling for RAM eligibility.

`ait init` records a supported root when one is already detected. On macOS, new-Task placement can still try the built-in `/Volumes/AIT_RAM` specification when the repository has no stored memory root. There is no environment-variable override for this Task-worktree contract.

`main-seed` fan-out

`main-seed` is a read-only materialization of one exact Snapshot at the default Line head. It is a cache, not a mutable shared checkout.

When a Task forks from the current default Line, bootstrap follows this path:

1. Resolve the verified memory root and Repository-specific worktree root.
2. Verify that `main-seed` matches the exact default-Line Snapshot.
3. Refresh or rebuild a missing, stale, or invalid seed from Snapshot authority.
4. Copy the seed into the new Task path.
5. Make only the copied Task tree writable and attach its Task, Change, and feature-Line metadata.
6. Expose the stable alias printed to the agent.

The per-file copy preference is:

1. macOS `clonefile`;
2. Linux `reflink`;
3. ordinary file copy when copy-on-write is unavailable.

Copy-on-write avoids paying for a complete independent physical copy at Task start. Each worktree still becomes logically independent as soon as an agent changes a file. If copying the seed fails safely, bootstrap removes the partial target and restores the feature-Line Snapshot directly. The Task is still isolated; only the acceleration path changed.

Keeping the seed current

After a successful Land to the default Line, AIT aligns `main-seed` with the landed Snapshot. The refresh path:

- uses a Repository and Line-specific refresh lock;
- validates the candidate worktree against the landed Snapshot;
- can promote the completed bound worktree or rebuild from Snapshot authority;
- prepares a staging seed and a recoverable previous-seed backup;
- makes the installed seed read-only; and
- rechecks the target Line generation before and during the atomic directory swap.

If `main` advances while the refresh is being prepared, AIT refuses to install the stale seed. A cache refresh failure does not manufacture successful seed state. It is reported separately so the landed code authority and the disposable acceleration cache are not confused.

Capacity and disk fallback

RAM parallelism must be bounded explicitly. By default, `task_worktree.main_seed_ram_max_bytes` is unset, so 1.1.1 has no automatic Snapshot-size cutoff. An operator can set the default-Line Snapshot size admitted to main-seed-backed RAM placement:

```
ait config set \
  --task-worktree-main-seed-ram-max-bytes <bytes>

ait config unset task-worktree-main-seed-ram-max-bytes
```

The placement decision for each **new** Task is:

```
configured seed limit exists AND Snapshot bytes > limit
  -> Repository-internal persistent disk
else if a supported RAM root can be selected and its Repository directory created
  -> RAM
else
  -> Repository-internal persistent disk
```

The comparison is strictly greater-than. A Snapshot exactly equal to the configured limit remains RAM-eligible. When the limit is exceeded, AIT records `main_seed_ram_budget_exceeded` and selects `.ait-worktree/<worktree-name>` under the persistent Repository root (normally SSD-backed). With the built-in unset value, this size-triggered switch is disabled.

AIT also selects that persistent root when every eligible memory candidate is unavailable or unusable. This includes no detected `tmpfs`/`ramfs` on Linux, no `DRIVE_RAMDISK` on Windows, an unavailable or failed RAM-volume attempt on macOS, a configured memory/ephemeral path that cannot be accepted under its declared root, or failure to create the Repository-specific worktree directory.

The read-only memory-root diagnostic reports a built-in minimum of zero bytes. 1.1.1 does not use that value as an automatic Task-placement switch and has no hidden low-memory percentage threshold. Operators that require a reserve must enforce it through host monitoring and stop admitting new Tasks before the filesystem is exhausted.

Placement does not migrate an existing Task midway from RAM to disk. If `main-seed` copying fails after the RAM path has been selected, AIT first tries a direct Snapshot restore at that same path; that is a materialization fallback, not an SSD switch. If the selected filesystem cannot complete the restore or later becomes full, the command fails and the operator must preserve recorded Snapshots, resolve capacity, and recreate or start work on an eligible root.

Fallback changes performance, not Task identity, Snapshot semantics, or Land safety. Inspect `root_source`, `ephemeral_enabled`, `target_path`, and `fallback_reason` in the Task/worktree output. `root_source` equal to `repo_internal_fallback` with `ephemeral_enabled: false` proves persistent-root placement; do not infer that a stable alias necessarily points to RAM.

Before starting a large fan-out, check current capacity and live worktrees:

```
ait doctor memory-root --json
ait worktree doctor --refresh --json
ait queue summary
```

Durability and recovery boundary

State	Survives loss of the RAM volume?	Recovery source
Task, Change, and Line authority	Yes	Persistent local authority or configured `ait-server`.
Recorded Snapshot content	Yes	Persistent Snapshot store and, when published, remote content authority.
`main-seed`	No requirement	Rebuild from the current default-Line Snapshot.
Clean Task worktree	No requirement	Recreate from its recorded feature-Line head.
Dirty edits not yet captured in a Snapshot	No	No automatic recovery guarantee.

This is the operational rule: create a Snapshot at coherent checkpoints when work must survive loss of volatile execution storage. `AIT_RAM` accelerates worktree materialization; it does not make unsnapshotted files durable.

After an unexpected unmount or restart, diagnose before deleting metadata:

```
ait doctor memory-root
ait worktree doctor --refresh --json
ait worktree recreate <worktree-name> --dry-run
ait worktree recreate <worktree-name>
```

If registry recovery must begin from a known Task and Change binding, use the explicit recovery surface shown by `ait worktree recover-task --help`. Recovery never reopens completed or canceled authority merely because an old directory exists.

Cleanup and operating rules

Successful Task closeout removes the bound worktree and archives its Task feature Line after verifying that the accepted Snapshot is still its head. This releases volatile capacity without deleting Snapshot history.

For leftovers after interruption, preview ownership-aware cleanup first:

```
ait worktree cleanup-candidates --json
ait worktree cleanup --dry-run
ait worktree prune-stale --dry-run
```

Keep these rules when operating many agents:

- Give every agent a distinct sprint item and Task.
- Enter only the `cd` path printed for that Task.
- Never use the canonical repository root as a shared editing directory.
- Record immutable checkpoints before relying on a volatile worktree.
- Let readiness refresh stale candidates; never bypass a base-head conflict.
- Treat CI as evidence for one selected Patchset, not for a Task name in general.
- Land through AIT so the target-head check and atomic transition remain in force.
- Diagnose and preview cleanup before removing a worktree.

What this does not guarantee

Worktree isolation prevents filesystem overwrites. It cannot prove that two textually compatible changes are behaviorally compatible. Two agents can edit different files and still change the same runtime contract. Repository-owned regression tests, review, and policy remain necessary.

AIT also does not provide unlimited RAM concurrency. Repository size, dirty deltas, tool caches, test processes, and runner capacity still determine a safe fan-out. `AIT_RAM` makes the cost visible and bounded; it does not remove the need for capacity planning.

Related

- `ait task`
- `ait snapshot`
- `ait workflow`
- `ait worktree` complete reference
- `ait doctor` complete reference

INFRASTRUCTURE AND INTEGRATIONS

Remote Infrastructure

Deploy the explicit ait-server and ait-runner boundary for remote authority, offsite recovery, and repository-owned CI.

AUDIENCE Operators

https://ait-native.dev/technical/v1.1.1/remote-infrastructure/

Activate remote authority only when needed

The normal local workflow requires no server. Remote infrastructure adds three separate capabilities when they solve a concrete need:

- remote management through shared, durable workflow authority;
- offsite preservation of recorded Snapshot and Line state for recovery; and
- repository-owned CI execution through typed Worker Jobs.

It also adds operator responsibility for authentication, TLS, network exposure, data-root backups, upgrades, and compatibility. Read `ait-server: Remote Authority and Recovery` for the complete data-flow, protection boundary, and recovery procedure.

Component boundary

- `ait-server` owns the repository registry, shared workflow state, policy, evidence, and Worker Job queue.
- `ait-runner` claims compatible typed jobs, materializes exact Snapshot content, executes the repository's declared CI entrypoint, and returns bounded results.

The runner does not become repository authority and does not choose a language-specific build system.

Evaluate the 1.1.1 server locally

Bind the evaluation container to loopback unless a reviewed secure ingress is already available.

```
docker network create ait-native
docker volume create ait-native-data
docker run --detach \
  --name ait-server \
  --network ait-native \
  --publish 127.0.0.1:8088:8088 \
  --restart unless-stopped \
  --volume ait-native-data:/var/lib/ait \
  ghcr.io/weita2026/ait-server:1.1.1

curl --fail http://127.0.0.1:8088/healthz
```

The public container index covers Linux `amd64` and `arm64`.

Register and configure the repository

`remote add` registers a numeric Repository authority when the local repository does not have one, then stores the remote. It verifies that exact authority instead of allocating another one when a repository index is already configured.

```
ait remote add origin <server-url> --default
ait remote list --json
ait config show --json
ait repo show --remote origin --json
```

Do not copy a repository index from another repository. The index is part of remote authority routing. Registration alone transfers no Snapshots; use the configured remote workflow or an explicit `ait push` to create a recoverable offsite copy of recorded state.

Runner contract

The runner accepts version-compatible jobs for one registered repository and executes its exact `ci/run.sh` or `ci/run.ps1` entrypoint. Source roots, attempt roots, worker identity, and repository index are operator-selected values.

Keep attempt storage isolated, monitor failed deliveries, and verify that attempt-owned data is cleaned after terminal results.

The dedicated `ait-runner: Native CI Execution Plane` chapter expands the exact 1.1.1 commands, Worker Job lifecycle, typed contracts, materialization limits, leases, evidence, deployment, and troubleshooting.

Readiness and closeout

Remote work publishes a selected Patchset and records completed CI, attestation, review, and policy evidence before final Task closeout.

```
ait workflow ready <change-id> --apply
ait workflow finish <change-id> --apply
```

The readiness and finish commands report the exact next action when a gate is pending. `ait task finish <task-or-change-id>` is the configured Task-level entry point when the selected remote Patchset is already ready.

Recovery is explicit

`ait-server` is a recovery source only for state that reached it. Dirty, untracked, and otherwise unrecorded workspace files are outside that boundary, and one server does not provide automatic failover. Keep the exact Repository index in the recovery inventory, protect the server data root independently, and rehearse the versioned recovery sequence in the detailed `ait-server` chapter.

INFRASTRUCTURE AND INTEGRATIONS

ait-server: Remote Authority and Recovery

Understand ait-server remote workflow authority, Snapshot preservation, disaster recovery, runner boundaries, and operating responsibilities.

AUDIENCE Repository owners and operators

https://ait-native.dev/technical/v1.1.1/ait-server/

Why run ait-server

A local AIT repository works without a server. Add `ait-server` when one or more of these outcomes are required:

- **Remote workflow management:** keep Repository, Plan, Task, Change, Patchset, evidence, policy, review, Land, and Worker Job state under one remote authority.
- **Offsite preservation and disaster recovery:** retain recorded Snapshot content and Line heads on a different machine or failure domain, then use them to rebuild local AIT authority and materialize source files.
- **Repository-owned CI:** admit typed jobs that a compatible `ait-runner` executes through the repository's declared `ci/run.sh` or `ci/run.ps1`.

These outcomes share the same server but remain distinct. A reachable server does not make every local file recoverable, and an offsite copy does not by itself provide automatic high availability or failover.

Protection starts with an AIT Snapshot. A dirty, untracked, ignored, or otherwise unrecorded workspace file does not reach the server merely because the repository is registered.

Authority owned by the server

One server data root owns a global append-only Repository registry. Every registered Repository receives a numeric `repository_index`; that index, not the display name, selects its authority. Repository names may repeat.

Each numeric Repository authority owns its remote:

- Snapshot metadata, content packs, ancestry, and Line heads;
- Plan, Task, Change, Patchset, attestation, review, policy, and Land records;
- repository-scoped Worker Jobs and CI result state; and
- protocol state needed to validate and sequence those operations.

The server is authoritative for those remote records. It does not own the developer's current working directory, choose the repository's test framework, or infer files that were never recorded.

Preserve the exact `repository_index` in the repository recovery inventory. It is routing metadata rather than a secret, but losing it removes the safe way to identify an existing authority when rebuilding a client. Never guess an index or substitute one from another Repository.

Register and verify a Repository

From an initialized repository, add the server as a remote:

```
ait remote add origin <server-url> --default
ait remote list --json
ait config show --json
ait repo show --remote origin --json
```

When no Repository index is configured, `remote add` registers a new numeric authority and persists the returned index. When an exact index is already configured, the command verifies that authority at the supplied server and fails closed on a mismatch.

Registration transfers no Snapshot or workspace content. It establishes the address and authority required by later workflow, push, pull, and recovery operations.

Remote workflow management

For a remote-scoped Task, the local Task worktree remains the authoring boundary. After a Snapshot is created, readiness publishes the selected Patchset and the Snapshot evidence required by the remote Change:

```
ait workflow ready <change-id> --apply
ait workflow finish <change-id> --apply
```

The server sequences CI, attestation, review, policy, and Land state. Final finish consumes the ready selected Patchset and advances the remote target Line. `workflow ready` and `workflow finish` are text-only decision surfaces; follow the exact next action they print when a gate is pending. The configured Task-level equivalent is `ait task finish <task-or-change-id>`.

Operators can inspect the registered authority and its execution state without changing it:

```
ait repo show --remote origin --json
ait repo jobs --remote origin --json
ait repo ci-capabilities --remote origin --json
```

Remote workflow publication is also an offsite copy of its recorded Snapshot content. A published but unlanded Patchset remains workflow evidence; it does not mean the remote target Line has advanced.

Preserve a Line explicitly

`ait push` is the direct synchronization path when the goal is to preserve an already-recorded Line independently of a governed remote Task:

```
ait status --json
ait snapshot create --message "Record the recovery point"
ait push --remote origin --line main
```

Push uploads the selected Line head and required Snapshot ancestry, then advances that remote Line. It does not capture dirty workspace files, create a Snapshot, publish a Patchset, merge divergent history, or perform Task finish. Push every Line that needs its own recovery point; registration does not start an automatic synchronization schedule.

The latest successfully landed or pushed remote Line head defines the recoverable point for that Line. Set the publication cadence from the maximum acceptable data-loss window, then verify the server and practice restoration instead of treating a successful registration as backup evidence.

Protection boundary

The server can preserve:

- Snapshot content and ancestry that were uploaded by push or remote workflow publication;
- remote Line heads that were successfully advanced;
- published remote workflow records and their admitted evidence; and
- repository-scoped CI and Worker Job state stored by that authority.

The server cannot recover:

- dirty, untracked, ignored, or un-Snapshotted workspace content;
- Snapshots or Line movement that remained local and were never pushed or published;
- files removed before the recorded recovery point;
- secrets, credentials, editor state, external services, or dependencies that are not part of the recorded Snapshot; or
- the server data root after the server itself is lost unless the operator also protected that data root elsewhere.

This boundary is why offsite preservation is deliberate: create the Snapshot, transfer it, confirm the remote state, and retain the Repository index needed to address it.

Restore when local authority is healthy

When `.ait` and the remote configuration are intact, pull the remote Line and materialize its head into a clean workspace:

```
ait status --json
ait diff --stat
ait pull --remote origin --line main --restore
```

Pull imports the remote Snapshot ancestry before classifying the local and remote heads. It fast-forwards when the remote is ahead, leaves the Line unchanged when local history is ahead, and refuses unrequested divergence. `--restore` replaces the workspace with the selected pulled head. A dirty workspace is rejected unless `--force` is explicitly supplied; preserve any surviving local-only files before making that choice.

Recover missing or damaged local authority

Use a separate clean recovery directory when the original `.ait` authority is missing or untrusted. Preserve the damaged directory for diagnosis instead of deleting it. Restore the trusted saved root routing fields as part of the local authority first: the exact `repository_index`, `default_remote`, and `remotes` map. 1.1.1 has no public `repository_index` setter, and `ait remote add` is a registration operation rather than a safe substitute for that recovery inventory.

```
# Run after the exact saved routing fields are restored.
ait config show --json
ait repo show --remote origin --json

ait remote recover-head --remote origin
ait remote recover-head --remote origin --apply
ait pull --remote origin --line main --restore
```

The first `recover-head` command is a preview. `--apply` downloads the verified Snapshot ancestry of the server's logical `main` head and reconstructs it inside a validated local Binary authority generation. 1.1.1 `recover-head` has no `--line` or multi-Line option. The final pull materializes `main` as source files. Recovery does not clone the server's remote workflow ledger into a separate local workflow ledger; remote Task, Change, Patchset, and job records remain queryable from their server authority.

Stop if the authority response does not match the saved index, the preview reports an unexpected head, validation fails, or the workspace contains files that must be preserved. If the exact saved routing fields cannot be restored, stop. Do not register replacement authority merely to bypass a mismatch. A complete retirement archive follows a different lifecycle: `ait repo restore --remote origin` creates restored authority with a new Repository index.

Runner and CI boundary

`ait-runner` is an executor, not durable Repository authority. It claims one compatible typed Worker Job, materializes exact Snapshot content into isolated attempt storage, runs the repository-authored entrypoint, and returns bounded results. The server owns admission, leases, job state, and result acceptance.

The runner does not select a language-specific build system, become a second copy of the Repository authority, or preserve arbitrary files from its attempt workspace. Keep attempt roots isolated, monitor failed result delivery, and clean attempt-owned data after terminal jobs.

See ait-runner: Native CI Execution Plane for the exact 1.1.1 commands, job lifecycle, request and result contracts, materialization limits, lease behavior, and deployment checks.

Protect the server data root

Using an offsite server protects the client only while the server authority survives. The container example stores it in the persistent volume mounted at `/var/lib/ait`; installed user and service modes use their configured data root. That complete root is the unit the operator must protect.

`ait-server` does not schedule a second replica, copy its own data root, elect a standby, or fail over automatically. Use an independent backup or storage-snapshot system in another failure domain. Capture an application-consistent copy by stopping writes or using storage that provides an equivalent consistency guarantee; do not treat an arbitrary live copy of

individual authority files as recoverable.

Retain the server version or immutable image digest, data-root backup, exact Repository index inventory, remote URL, and separately managed access material needed for a recovery drill. Validate a restored copy in isolation before it serves traffic:

```
ait-server probe --data <restored-data-root> --defer-ci-admission
```

Probe failure means the restored root must not be activated. A successful `/healthz` response proves that the running process is healthy; it does not prove backup recency, completeness, or recoverability.

Operational checklist

- Bind the service to loopback or a reviewed private ingress by default.
- Terminate TLS and enforce caller verification, network controls, monitoring, and capacity limits at the trusted ingress and host boundary.
- Pin compatible 1.1.1 client, server, and runner artifacts.
- Record every Repository's numeric index outside the client data root.
- Define and verify a Snapshot publication cadence for each protected Line.
- Back up the complete server data root to another failure domain.
- Rehearse `probe`, `recover-head`, and `pull --restore` with an isolated copy.
- Monitor server health, failed jobs, runner compatibility, and storage growth.

See the complete `ait-server` REST API Reference for every 1.1.1 HTTP route, request contract, response, and error boundary. Remote Infrastructure has the container boundary, `ait-runner: Native CI Execution Plane` has the full runner operations contract, and Troubleshooting has the general evidence-preservation rules.

INFRASTRUCTURE AND INTEGRATIONS

ait-runner: Native CI Execution Plane

Operate the 1.1.1 native runner through exact Worker Job claims, Snapshot materialization, isolated attempts, bounded evidence, leases, and deployment controls.

AUDIENCE CI operators, Repository owners, and integrators

https://ait-native.dev/technical/v1.1.1/ait-runner/

What ait-runner does

`ait-runner` is the native execution plane for CI work admitted by `ait-server`. The server owns the Repository authority, Worker Job record, lease, state transition, and accepted result. The runner owns one bounded execution attempt: it claims compatible work, reconstructs the exact recorded source, invokes the Repository's CI entrypoint, returns evidence, and removes its attempt directory.

The runner is useful when CI must execute away from the developer worktree, when different machines serve different Repository sets, or when an operator must replay one known Worker Job pair. It is not a Repository backup, a policy engine, a scheduler independent of `ait-server`, or a language-specific build system.

Only two Binary Worker Job kinds are externally claimable in 1.1.1:

Fixed kind	Public name	Execution purpose
7	`patchset.ci`	Validate the selected Patchset Snapshot for governed readiness.
11	`repo.ci`	Run Repository CI against one exact recorded Snapshot.

The durable job identity is the pair `(repository_index, worker_job_index)`. Never substitute a Repository name for that pair when diagnosing or replaying a Binary Worker Job.

Four operating situations

Persistent execution service

Run `serve` under a service manager when one worker should poll continuously, maintain leases, execute compatible jobs, and deliver terminal results:

```
ait-runner serve \
  --server https://ait.example.internal \
  --worker-id runner-taipei-01 \
  --attempt-root /srv/ait/runner-attempts
```

`--worker-id` is required operational identity, not an environment setting. The server URL, source root, and attempt root are also explicit command arguments so the launched process describes its operating boundary.

Repository-scoped worker pool

Repeat `--repository-index` to restrict one service instance to an exact set of Repository authorities. Duplicate indexes are normalized:

```
ait-runner serve \
  --server https://ait.example.internal \
  --worker-id runner-linux-arm64-02 \
  --repository-index 3 \
  --repository-index 8 \
  --attempt-root /srv/ait/runner-attempts
```

Omitting the filter lets the compatible queue select across registered Repositories. Use filters to express an intentional fleet boundary, not to guess which Repository owns a failing job.

One exact Worker Job

Use `run-job` for a known Binary pair during controlled diagnosis or recovery. It claims, executes, heartbeats, and finishes only that pair:

```
ait-runner run-job \
  --server https://ait.example.internal \
  --repository-index 3 \
  --worker-job-index 42 \
  --attempt-root /srv/ait/runner-attempts
```

The server must advertise the current `ait.runner.native-job.v3` contract. The command fails if the pair is not claimable, its lease proof is invalid, or the request does not match the claimed Repository.

Local request validation

Use `execute` to exercise one typed local-directory request without claiming a server job. This is useful for validating the runner boundary itself; it does not create or update a Worker Job:

```
{
  "contract": "ait.runner.native-job.v3",
  "label": "local-repository-ci",
  "source": {"kind": "local_directory", "path": "."},
  "command": {
    "argv": ["./ci/run"],
    "working_directory": ".",
    "environment": {}
  },
  "timeout_ms": 900000,
  "suite_id": "repository-ci"
}
```

```
ait-runner execute \
  --request request.json \
  --source-root /srv/ait/sources \
  --attempt-root /srv/ait/runner-attempts
```

`--request -` reads the same JSON from standard input and is the default. A server-provided `remote_snapshot` request requires the server materialization provider used by `run-job` or `serve`; standalone `execute` is the local source path.

Complete 1.1.1 command surface

```
ait-runner doctor --server <SERVER>

ait-runner execute \
  [--request <REQUEST>] \
  [--source-root <SOURCE_ROOT>] \
  [--attempt-root <ATTEMPT_ROOT>]

ait-runner run-job \
  --server <SERVER> \
  --repository-index <REPOSITORY_INDEX> \
  --worker-job-index <WORKER_JOB_INDEX> \
  [--source-root <SOURCE_ROOT>] \
  [--attempt-root <ATTEMPT_ROOT>]

ait-runner serve \
  --server <SERVER> \
  --worker-id <WORKER_ID> \
  [--repository-index <REPOSITORY_INDEX>]... \
  [--once] \
  [--poll-interval-ms <MILLISECONDS>] \
  [--heartbeat-interval-ms <MILLISECONDS>] \
  [--source-root <SOURCE_ROOT>] \
  [--attempt-root <ATTEMPT_ROOT>]
```

Command	Exact behavior
`doctor`	Reads live server health and negotiates the runner contract without claiming a job.
`execute`	Parses one bounded v3 request and executes its local source in an isolated attempt.
`run-job`	Claims and completes one exact Binary Worker Job pair; current v3 support is required.
`serve`	Polls continuously or, with `--once`, exits after one idle check, one delivered job, or one error.

`execute`, `run-job`, and `serve` default `--source-root` to `.`. An omitted `--attempt-root` uses the platform temporary directory below `ait-runner/attempts`. `serve` defaults to a 1,000 ms poll interval and a 30,000 ms requested heartbeat interval. Both intervals must be nonzero; the effective Binary heartbeat is further bounded by the active lease.

End-to-end job lifecycle

1. A remote workflow or Repository CI request causes `ait-server` to commit a typed Worker Job for one Repository.
2. The runner checks server health and selects the current Binary runner contract. `doctor` stops here and claims nothing.
3. `serve` claims the next compatible job, or `run-job` claims the exact `(repository_index, worker_job_index)` pair. The server returns a lease proof, attempt count, fixed job kind, and typed runtime request.
4. The runner validates the proof, job kind, Repository index, request size, paths, arguments, environment, platform, and CI entrypoint before execution.
5. For `remote_snapshot`, it downloads and verifies the exact Snapshot Tree, Blob packs, and locked external Repository Snapshots into a new attempt.
6. The logical argv selector `./ci/run` resolves to `ci/run.sh` on Unix or `ci/run.ps1` on Windows. The runner starts it with direct argv, not a concatenated shell command.
7. A separate heartbeat maintains the server lease while the process runs. A timeout terminates the owned process group before evidence is finalized.
8. The runner reduces stdout and stderr to bounded evidence, records exit and materialization facts, removes the attempt directory, and validates the terminal result size.
9. The runner submits complete or fail with the original lease proof. The server alone decides whether that transition is still admissible and stores the accepted terminal state.

The selected Snapshot remains immutable throughout the attempt. A runner never executes dirty developer-worktree content for a remote Binary job.

Request contract

The current request schema is `ait.runner.native-job.v3`. Unknown fields are rejected. Its top-level fields are:

Field	Contract
`contract`	Required exact string `ait.runner.native-job.v3`.
`label`	Optional nonempty label, at most 256 bytes.
`source`	Required `local_directory` or current `remote_snapshot` source object.
`command`	Required direct-argv command object.
`timeout_ms`	Optional; defaults to 900,000 ms and must be 1 through 86,400,000 ms.
`suite_id`	Optional nonempty suite label, at most 256 bytes.

For a server-delivered source, `remote_snapshot` carries exact `repository_index`, `repository_name`, `snapshot_id`, and a map of external Repository names to their numeric indexes. The request's Repository index must match the claimed Worker Job pair. The command object requires `argv`, while `working_directory` defaults to `.` and `environment` defaults to an empty map.

The first argv value is always the logical `./ci/run` selector. Additional values are passed directly to the platform entrypoint. The working directory must remain relative and confined to the materialized workspace.

Materialization and attempt isolation

Every execution uses a uniquely named `attempt-*` directory containing the workspace, downloaded packs, and temporary logs. `serve` takes an exclusive lock on its configured attempt parent and reclaims only exact runner-owned stale attempt directories. Cleanup handles read-only files and does not follow symlinks outside the attempt.

Remote materialization fails closed on malformed inventory, checksum mismatch, unknown or excessive records, Tree cycles or depth, path escape, symlink source or entrypoint, and content that exceeds the declared bounds. Locked external Repositories are materialized from their own exact Snapshots, not from an ambient checkout.

The child CI process receives these runner-owned values:

Name	Meaning
`AIT_RUNNER_ATTEMPT_ROOT`	Exact root of the active runner-owned attempt.
`AIT_RUNNER_WORKSPACE`	Exact materialized primary Repository workspace.
`AIT_EXTERNAL_<NAME>_REPO_ROOT`	Exact materialized root for one normalized locked external Repository.

These are injected process boundaries, not settings for launching the runner. The only supported runner credential environment variable is the optional `AIT_SERVER_TOKEN`. Server URL and worker identity remain `--server` and `--worker-id`; `AIT_SERVER_URL` and `AIT_RUNNER_WORKER_ID` are not inputs.

Bounded execution and evidence

Boundary	1.1.1 limit
Typed request	1 MiB.
Terminal result	64 KiB after JSON encoding.
Timeout	15 minutes by default; 24 hours maximum.
Command argv	256 values; 16 KiB each; 128 KiB total.
Command environment	256 entries; 256 KiB total key and value bytes.
Stdout and stderr evidence	8 KiB tail for each stream.

Each stream record contains the total byte count, SHA-256 of all captured bytes, base64 tail, tail byte count, and a truncation flag. The full temporary log is removed with the attempt, so terminal evidence is intentionally bounded rather than a hidden log archive.

Remote Snapshot import also enforces a 64 MiB manifest limit, 10,000,000 materialized files, 512 GiB materialized bytes, a 16 GiB per-pack download limit, and eight concurrent pack download/decode operations. These are safety ceilings, not recommended job sizes; operators should set substantially lower capacity expectations for a real worker host.

Result and diagnostic contracts

Successful command execution writes one `ait.runner.native-result.v1` object. Its terminal `status` is `succeeded`, `command_failed`, or `timed_out`, and `tests_status` is `pass` only for `succeeded`. The result contains suite records, exit code or signal, duration, materialized file and byte counts, bounded stdout and stderr evidence, and cleanup evidence.

Server-facing commands wrap an accepted terminal transition in `ait.runner.delivery.v1`. Other public output contracts are:

Contract	When emitted
`ait.runner.doctor.v1`	`doctor` confirms health and reports the selected runner contract.
`ait.runner.service.v1`	`serve --once` finds no compatible job and returns `idle`.
`ait.runner.serve-event.v1`	Persistent `serve` reports a bounded job failure to stderr and continues polling.
`ait.runner.error.v1`	A command fails before it can return its normal result.

`--once` is fail-fast and returns after its single outcome. Persistent `serve` continues after a bounded per-job failure event. When polling cannot reach the server, persistent service mode uses bounded reconnect backoff up to about 30 seconds. Once the current Binary contract has been selected, the process will not silently downgrade. A heartbeat failure fails the claimed job; the runner does not overlap ambiguous heartbeat retries.

Deployment

1.1.1 publishes native runner executables for macOS, Linux/glibc, and Windows on both arm64 and x86_64. The Linux OCI index covers `amd64` and `arm64`:

```
ghcr.io/weita2026/ait-runner:1.1.1
```

The image entrypoint is `ait-runner`, its working directory is `/workspace`, and it runs as numeric UID/GID 65532. Give the attempt path a writable volume owned for that identity; a remote Snapshot worker does not need a mutable local Repository checkout:

```
docker volume create ait-runner-attempts
docker run --rm \
  --network ait-native \
  --volume ait-runner-attempts:/var/lib/ait-runner \
  ghcr.io/weita2026/ait-runner:1.1.1 \
  serve \
  --server http://ait-server:8088 \
  --worker-id runner-container-01 \
  --repository-index 0 \
  --attempt-root /var/lib/ait-runner \
  --once
```

Use a service or container secret for `AIT_SERVER_TOKEN`; do not embed the token in the image, command, Repository, or retained logs. Pin the immutable 1.1.1 image digest in production after verifying the published index.

Security and operating checklist

- Run each worker under a dedicated OS identity or container with least filesystem and network access.
- Restrict server ingress, use authenticated TLS at the trusted boundary, and keep `AIT_SERVER_TOKEN` out of argv and files under the attempt root.
- Put attempt storage outside Repository authority and outside any source root.
- Do not share one attempt root between concurrently running `serve` processes; the exclusive lease will reject it.
- Author `ci/run.sh` and `ci/run.ps1` as the Repository's reviewed CI boundary. Do not expect the runner to infer a package manager or test command.
- Monitor server Worker Job state, runner failure events, lease loss, disk capacity, cleanup evidence, and version-contract mismatches.
- Treat `command_failed` as repository CI evidence, `timed_out` as an execution limit, and `ait.runner.error.v1` as a runner or request-boundary failure.

Start diagnosis without claiming work:

```
ait-runner --version
ait-runner doctor --server https://ait.example.internal
ait repo jobs --remote origin --json
ait repo ci-capabilities --remote origin --json
```

Then compare the exact Repository and Worker Job indexes, selected runner contract, attempt count, lease timing, terminal operation, stream digests, and cleanup evidence. Preserve server-side job evidence; do not preserve or reuse a partially cleaned attempt directory as if it were authoritative source.

For the HTTP claim, heartbeat, complete, fail, Snapshot import, and pack routes, see the ait-server REST API Reference. For server authority and offsite recovery, see ait-server: Remote Authority and Recovery. The complete public environment registry is in Appendix: Environment Variables.

INFRASTRUCTURE AND INTEGRATIONS

ait-server REST API Reference

Call every 1.1.1 ait-server HTTP operation with implementation-aligned request fields, responses, errors, recovery flows, and security boundaries.

AUDIENCE Integrators, runner authors, and operators

https://ait-native.dev/technical/v1.1.1/ait-server/rest-api/

What this reference covers

This is the complete HTTP surface exposed by the 1.1.1 `ait-server` release router. It accounts for **51 route templates**, **71 HTTP method bindings**, and **78 callable operations** after the action suffixes accepted by shared routes are expanded.

The API is JSON over HTTP with resource reads and writes plus explicit action endpoints such as `:runCi`, `:close`, and `:submit`. Those action endpoints are intentional; the surface is not a strictly uniform CRUD API.

This chapter documents the server interface used by `ait`, `ait-runner`, and recovery tooling. Normal users should prefer the `ait` command guides because the clients enforce sequencing and compatibility checks that a raw HTTP caller must implement itself.

Security and transport boundary

Treat the 1.1.1 listener as a trusted-network service boundary. Do not expose the bare listener to the public Internet. Keep it on loopback or a reviewed private network and terminate TLS, caller verification, request limits, and access policy at a trusted ingress. The default listener is `127.0.0.1:8088`; choose a different bind address with the explicit server command option.

The restore token and Worker Job lease token are operational capabilities, not a substitute for ingress authentication. Do not log or disclose them.

For the examples below:

```
BASE=http://127.0.0.1:8088
REPOSITORY_INDEX=17
```

Common protocol rules

Addressing and identifiers

- `{repository_index}` and `{worker_job_index}` are canonical unsigned base-10 `u32` values. `0` is valid; `01`, signs, spaces, and values above `4294967295` are rejected.
- Repository names are display and discovery data. Names can repeat and do not select authority; the numeric Repository index does.
- Task, Change, Patchset, Snapshot, Land, Plan, and revision identifiers are opaque public identifiers. Preserve their spelling and URL-encode path segments when necessary.
- Time fields ending in `_at_s` are unsigned Unix seconds. Fields such as `created_at` are RFC 3339 strings. A missing event is normally `null` in a projection rather than an invented timestamp.

Request headers and body limits

Use `Content-Type: application/json` for JSON requests and `Accept: application/json` for JSON responses. The standard JSON extractor applies the framework's normal 2 MiB body ceiling. The following bulk routes raise the request ceiling to **2 GiB**:

- zstd bulk plan, commit, and pull-manifest requests;
- zstd Object Pack and Tree Pack transfer routes; and
- Repository restore file upload.

Raw pack and recovery-file bodies require an exact media type:

Body	Required `Content-Type`	Response type
Object Pack	`application/vnd.ait.remote-sync.object-pack+zstd`	same value
Tree Pack	`application/vnd.ait.remote-sync.tree-pack+zstd`	same value
Retirement or restore file	`application/vnd.ait.remote-authority-file.v1`	same value

Empty Object Pack and Tree Pack uploads are rejected. A request beyond the active body limit is rejected before the handler runs.

Success and error responses

Most successful operations return `200 OK` with a route-specific JSON value. Repository registration returns `201 Created` when it appends a new authority and `200 OK` when it returns the matching existing registration. Starting a Repository restore session returns `201 Created`.

Server-generated application errors use this stable body:

```
{"error":"human-readable diagnostic"}
```

Status	Meaning
`200`	Successful read, idempotent replay, update, action, or binary download.
`201`	New Repository authority or restore session created.
`400`	Invalid path value, request field, state transition, or malformed JSON handled by the route.
`404`	Unknown route, Repository, entity, pack, file, or action suffix.
`405`	The route exists but the HTTP method is not bound.
`409`	Authority conflict, lifecycle conflict, stale Line head, governed target-Line movement, or mismatched retirement acknowledgement.

Status	Meaning
`413`	Request body exceeds the active route limit.
`415`	A JSON extractor or binary route rejects the request media type.
`422`	JSON syntax was accepted but could not be deserialized into the typed request.
`500`	Corrupt, incompatible, I/O-failed, or otherwise internal authority operation.
`503`	Startup is incomplete or an authority is retryably busy. Application-generated `503` errors include `Retry-After: 1`.

Do not parse the diagnostic text as a long-term machine contract. Branch on the HTTP status and the documented response fields; keep the text for operator diagnosis.

Startup behavior

The listener is available before all serving authority has finished opening. Until activation:

- `GET /healthz` returns `503` and `{"ready":false,"status":"starting"}`;
- every other path returns `503` and `{"error":"ait-server is starting","ready":false,"status":"starting"}`.

The ready router replaces the startup proxy atomically. After activation, `GET /healthz` returns `200` with `ready: true`.

Repository lifecycle admission

`GET`, `HEAD`, and `OPTIONS` requests pass lifecycle admission. A mutation under `/v1/native/repository-authorities/{repository_index}/...` normally requires an active Repository.

Retirement status/start, retirement purge, retirement abort, and the Worker Job `:claim`, `:heartbeat`, `:complete`, and `:fail` drain actions remain available while a Repository is retiring. Other mutations against a retiring or purged Repository are rejected. This lets in-flight work drain and backup or restore procedures finish without admitting new workflow work.

Complete operation index

The compact `endpoints` array returned by the handshake is a discovery hint, not the complete route list. The tables below are the complete 1.1.1 release surface.

Service discovery

Method and path	Operation
`GET /healthz`	Read readiness and authority backend.
`GET /v1/handshake`	Read protocol, package, CI, runner, and compact endpoint capabilities.
`GET /v1/native/capabilities`	Read operational Repository and Worker Job contracts.

Repository registry, backup, restore, and jobs

Method and path	Operation
`GET /v1/native/repository-authorities`	List or discover Repository authorities.
`POST /v1/native/repository-authorities`	Register a Repository authority.
`GET /v1/native/repository-authorities/{repository_index}`	Read one Repository authority.
`POST /v1/native/repository-authorities/{repository_index}:runCi`	Enqueue Repository CI.
`GET /v1/native/repository-authorities/{repository_index}/retirement`	Read retirement drain/export status.
`POST /v1/native/repository-authorities/{repository_index}/retirement`	Begin retirement.

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/retirement/files/{file_path}`	Download one manifest-listed authority file.
`POST /v1/native/repository-authorities/{repository_index}/retirement/purge`	Purge after exact export acknowledgement.
`POST /v1/native/repository-authorities/{repository_index}/retirement/abort`	Return a retiring Repository to active state.
`POST /v1/native/repository-restores`	Begin a restore session.
`PUT /v1/native/repository-restores/{restore_token}/files/{file_path}`	Upload one manifest-listed restore file.
`POST /v1/native/repository-restores/{restore_token}/commit`	Validate and activate a restored Repository.

Method and path	Operation
`POST /v1/native/worker-jobs:claim`	Claim the next matching external-runner job.
`GET /v1/native/repository-authorities/{repository_index}/worker-jobs`	List Repository Worker Jobs.
`GET /v1/native/repository-authorities/{repository_index}/worker-jobs/{worker_job_index}`	Read one Worker Job.
`POST /v1/native/repository-authorities/{repository_index}/worker-jobs/{worker_job_index}:claim`	Claim one exact job.
`POST /v1/native/repository-authorities/{repository_index}/worker-jobs/{worker_job_index}:heartbeat`	Extend an active lease.
`POST /v1/native/repository-authorities/{repository_index}/worker-jobs/{worker_job_index}:complete`	Commit a successful attempt.

Method and path	Operation
`POST /v1/native/repository-authorities/{repository_index}/worker-jobs/{worker_job_index}:fail`	Record a retryable or terminal attempt failure.

Lines, Snapshots, and remote sync

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/lines`	List Lines.
`GET /v1/native/repository-authorities/{repository_index}/lines/{line_name}`	Read one Line.
`PUT /v1/native/repository-authorities/{repository_index}/lines/{line_name}`	Create or update a Line with compare-and-set.
`POST /v1/native/repository-authorities/{repository_index}/lines/{line_name}:close`	Close a Line.
`POST /v1/native/repository-authorities/{repository_index}/snapshots:exists`	Test Snapshot presence in bulk.
`GET /v1/native/repository-authorities/{repository_index}/snapshots/{snapshot_id}`	Export Snapshot metadata and optionally content.

Method and path	Operation
`POST /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/plan`	Classify present and missing Snapshots and packs.
`POST /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/commit`	Commit uploaded pack metadata, locators, Snapshots, and optional Line update.
`GET /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/import-manifests/{snapshot_id}`	Get one Snapshot import closure.
`POST /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/pull-manifests`	Get a bounded ancestry import closure.
`GET /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/object-packs/{pack_id}`	Download one Object Pack.
`PUT /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/object-packs/{pack_id}`	Upload one Object Pack.

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/tree-packs/{pack_id}`	Download one Tree Pack.
`PUT /v1/native/repository-authorities/{repository_index}/remote-sync/zstd-bulk/tree-packs/{pack_id}`	Upload one Tree Pack.

Task and queue read models

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/tasks`	List Tasks, newest first.
`POST /v1/native/repository-authorities/{repository_index}/tasks`	Create a Task, optionally Plan-bound.
`POST /v1/native/repository-authorities/{repository_index}/task-start`	Atomically create/revise/select a Plan item, Task, and first Change.
`GET /v1/native/repository-authorities/{repository_index}/tasks/{task_ref}`	Read one Task.
`POST /v1/native/repository-authorities/{repository_index}/tasks/{task_ref}:close`	Complete or abandon a Task.
`GET /v1/native/repository-authorities/{repository_index}/read/tasks/{task_id}/audit`	Read Task closeout evidence.

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/read/queue-summary`	Read the queue summary projection.
`GET /v1/native/repository-authorities/{repository_index}/read/task-queue`	Read the Task queue projection.
`GET /v1/native/repository-authorities/{repository_index}/read/reviewer-inbox`	Read review work assigned to the Repository.

Change, Patchset, review, policy, CI, and Land

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/changes`	List Changes, newest first.
`POST /v1/native/repository-authorities/{repository_index}/changes`	Create a Change under a Task.
`GET /v1/native/repository-authorities/{repository_index}/changes/{change_ref}`	Read one Change.
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}:close`	Archive, abandon, or supersede a Change.
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}:selectPatchset`	Select a Patchset.
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}:requestReview`	Request one reviewer group.

Method and path	Operation
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}:submit`	Submit direct Land for a Change.
`GET /v1/native/repository-authorities/{repository_index}/changes/{change_ref}/patchsets`	List Patchsets in ordinal order.
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}/patchsets`	Publish a Patchset.
`GET /v1/native/repository-authorities/{repository_index}/changes/{change_ref}/reviews`	Read review rows and summary counts.
`POST /v1/native/repository-authorities/{repository_index}/changes/{change_ref}/reviews`	Record a review action.
`GET /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}`	Read one Patchset and its CI projection.

Method and path	Operation
`POST /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}:runCi`	Start or reuse Patchset CI and enqueue a Worker Job when required.
`POST /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}:evaluatePolicy`	Evaluate current attestation, CI, and review gates.
`GET /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}/attestation`	Read the latest attestation.
`PUT /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}/attestation`	Append a new attestation.
`GET /v1/native/repository-authorities/{repository_index}/patchsets/{patchset_id}/policy`	Read the latest policy decision or pending projection.
`GET /v1/native/repository-authorities/{repository_index}/read/patchsets/{patchset_id}/ci-status`	Read compact CI readiness plus recent jobs.

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/lands/{submission_id}`	Read one Land attempt.
`POST /v1/native/repository-authorities/{repository_index}/task-land`	Atomically resolve and Land one Task or exact Change.
`POST /v1/native/repository-authorities/{repository_index}/history-promotion:prepare`	Prepare recorded local-land history for governed remote Land.

Sprint and Plan authority

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/sprints`	List Plans, optionally by artifact path.
`POST /v1/native/repository-authorities/{repository_index}/sprints`	Create a Plan and first revision.
`GET /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}`	Read Plan detail and head revision.
`PATCH /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}`	Change Plan status.
`GET /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}/revisions`	List Plan revisions newest first.
`POST /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}/revisions`	Append a revision with optional head compare-and-set.

Method and path	Operation
`GET /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}/revisions/{plan_revision_id}`	Read one full revision.
`GET /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}/revisions/{plan_revision_id}/artifacts`	Read the same full revision, including its packed artifact projection.
`PUT /v1/native/repository-authorities/{repository_index}/sprints/{plan_id}/revisions/{plan_revision_id}/artifacts`	Reserved route; 1.1.1 rejects supporting artifact writes.
`POST /v1/native/repository-authorities/{repository_index}/sprint-plan-ids/by-contains`	Find active Plan IDs matching all normalized terms.
`POST /v1/native/repository-authorities/{repository_index}/sprint-task-linkage/resolve`	Normalize and validate Task-to-Plan linkage.
`GET /v1/native/repository-authorities/{repository_index}/read/plans/candidate-inputs`	Read matching Plans and their linked Task candidates.

Discovery endpoints

Health

```
curl --fail-with-body "$BASE/healthz"
```

Ready response:

```
{
  "ready": true,
  "authority_backend": "binary_v0",
  "repository_identity": "repository_index",
  "operational_capabilities": {
    "contract": "ait.server.operational-capabilities.v1",
    "repository_identity": "binary-repository-index.v0",
    "worker_job_identity": "binary-worker-job-key.v0",
    "runner_contracts": ["ait.runner.native-job.v3"]
  }
}
```

Treat HTTP `200` plus `ready: true` as process readiness. It does not prove that a backup is current or recoverable.

Handshake

```
curl --fail-with-body "$BASE/v1/handshake"
```

The response fields are:

Field	Meaning
`ready`	Ready-router state.
`contract_version`	Agent/server protocol compatibility version.
`package_version`	Running `ait-server` package version.
`authority_backend`	`binary_v0` in 1.1.1.
`repository_identity`	`repository_index`.
`ci_capabilities`	Remote-sync features and native runner contract.

Field	Meaning
`operational_capabilities`	Repository and Worker Job identity contracts.
`supported_async_job_types`	Ten durable job type names known by the server.
`endpoints`	Compact discovery subset; not the complete operation index.

The native runner capability declares `ait.runner.native-job.v3`, result contract `ait.runner.native-result.v1`, queue contract `ait.server.worker-job.service.v1`, remote Snapshot input, direct argument execution, and the platform `ci/run.sh` or `ci/run.ps1` entrypoint.

Operational capabilities

`GET /v1/native/capabilities` returns the `operational_capabilities` object without the larger handshake payload. Use it before writing a direct Worker Job client.

Repository registration and Repository CI

Register an authority

```
curl --fail-with-body \
  -H 'Content-Type: application/json' \
  -d '{"repository_name":"ait-native-site","namespace":"w","policy_flags":0}' \
  "$BASE/v1/native/repository-authorities"
```

Request field	Required	Rules
`repository_name`	yes	Non-empty UTF-8 display name; duplicates are allowed.
`namespace`	yes	Zero, one, or two ASCII letters, digits, _, or -. The server stores exact zero-padded bytes.
`policy_flags`	yes	Unsigned u8.

Unknown fields are rejected. A non-empty namespace already owned by a live Repository is replayed only when name and policy flags also match; otherwise registration conflicts.

```
{
  "contract": "ait.server.repository-registration.v1",
  "created": true,
  "repository": {
    "repository_index": 17,
    "repository_name": "ait-native-site",
    "lifecycle_kind": 1,
    "namespace": "w",
    "policy_flags": 0,
    "created_at_s": 1786800000,
    "updated_at_s": 1786800000,
    "tombstoned": false
  }
}
```

Persist `repository_index` from the response. Do not rediscover by name and guess when several authorities share a display name.

Discover and read authorities

```
curl --get --fail-with-body \
  --data-urlencode 'repository_name=ait-native-site' \
  "$BASE/v1/native/repository-authorities"

curl --fail-with-body \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX"
```

The list response contains `contract`, `repositories`, `count`, and a `discovery` object. `discovery.routing_authority` is `false`: a name filter is for inventory, not route selection. The item response wraps

one `repository` under `ait.server.repository-authority.v1`.

Lifecycle values are `1` active, `2` retiring, and `3` purged. Purged entries are omitted from list discovery.

Enqueue Repository CI

The route uses the action suffix on the Repository segment:

```
curl --fail-with-body \
  -H 'Content-Type: application/json' \
  -d '{"snapshot_id":"SNP-EXAMPLE"}' \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX:runCi"
```

`snapshot_id` is optional. When absent, the server selects logical `main`'s current head. The request fails when neither choice resolves to a Snapshot. The response contains `repository_index`, `snapshot_id`, physical `snapshot_index`, `queued`, the Worker Job projection, and `delivery: "binary_worker_job"`.

Repository retirement, export, and restore

Retirement is the API-level backup/export sequence for one complete numeric Repository authority. It is deliberately staged so new mutations stop, Worker Jobs drain, every exported file is verified, and purge requires an exact acknowledgement.

1. Begin and poll retirement

These POST operations have no request body:

```
curl --fail-with-body -X POST \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/retirement"

curl --fail-with-body \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/retirement"
```

The response contains:

```
{
  "contract": "ait.server.repository-retirement.v1",
  "repository": {"repository_index":17,"lifecycle_kind":2},
  "drain": {"queued_worker_jobs":0,"running_worker_jobs":0},
  "ready_for_export": true,
  "manifest": {
    "schema": "ait.remote-export.v1",
    "state": "complete",
    "repo_name": "ait-native-site",
    "namespace": "w",
    "exported_at_s": 1786800100,
    "files": [
      {"path":"...","size":1234,"sha256":"..."}
    ]
  }
}
```

`manifest` is `null` until queued and running Worker Jobs both reach zero. Only `ready_for_export: true` is permission to fetch the listed files.

2. Download and verify every file

For every `manifest.files[]` entry, URL-encode its relative `path` and fetch:

```
curl --fail-with-body \
  -o authority-file.bin \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/retirement/files/<encoded-path>"
```

The response is raw `application/vnd.ait.remote-authority-file.v1`. Verify both `size` and lowercase hexadecimal `sha256` against the manifest. A path not present in the current manifest returns `404`.

3. Purge only after durable export

POST the **entire unchanged manifest object** as the purge body:

```
curl --fail-with-body \
  -H 'Content-Type: application/json' \
  --data-binary @remote-export-manifest.json \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/retirement/purge"
```

The server recomputes its current complete export and requires exact equality. It rejects an incomplete acknowledgement, a changed file inventory, or undrained jobs. Success returns `contract`, `purged`, `already_purged`, and the terminal Repository projection. Replaying the same valid acknowledgement is idempotent.

Abort retirement

Before purge, this no-body operation returns the Repository to active state:

```
curl --fail-with-body -X POST \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/retirement/abort"
```

The response reports `aborted: true` and `already_aborted`. A purged authority cannot be reactivated.

Restore an exported authority

Start a session with the verified manifest plus the `u8` policy flags for the new registry entry:

```
curl --fail-with-body \
  -H 'Content-Type: application/json' \
  -d '{"manifest":<complete-manifest>,"policy_flags":0}' \
  "$BASE/v1/native/repository-restores"
```

The typed request accepts exactly `manifest` and `policy_flags`. The manifest must have:

Field	Rule
`schema`	`ait.remote-export.v1`
`state`	`complete`
`repo_name`	Non-empty source Repository name.
`namespace`	Exact exported namespace. It must not be owned by a live Repository.
`exported_at_s`	Export time in Unix seconds.
`files`	Exact array of `{path, size, sha256}` entries.

The `201` response returns a random 32-character hexadecimal `restore_token` and `state: "uploading"`. At most 64 uncommitted sessions are admitted.

Upload each listed file with its exact relative path and media type:

```
curl --fail-with-body -X PUT \
  -H 'Content-Type: application/vnd.ait.remote-authority-file.v1' \
  --data-binary @authority-file.bin \
  "$BASE/v1/native/repository-restores/$RESTORE_TOKEN/files/<encoded-path>"
```

Each upload is checked against the manifest size and digest. Then commit with no request body:

```
curl --fail-with-body -X POST \
  "$BASE/v1/native/repository-restores/$RESTORE_TOKEN/commit"
```

Commit validates the full staged archive, appends a new numeric Repository entry, normalizes the restored authority to that new index, and returns `ait.server.repository-restore.v1`. Replaying commit on the same live session returns the recorded response. Restore does not preserve the old numeric index; the response's new index is authoritative.

Worker Job API

The server knows ten durable job kinds, but the external 1.1.1 runner may claim only `7` (`patchset.ci`) and `11` (`repo.ci`). Other kinds are server-side operations and are rejected by the external claim endpoints.

Worker Job states are `1` queued, `2` running, `3` succeeded, and `4` failed. The pair `(repository_index, worker_job_index)` is the public identity.

List and inspect

```
curl --get --fail-with-body \
  --data 'state_kind=1' \
  --data 'limit=50' \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/worker-jobs"
```

`state_kind` is optional and must be `1..4`. `limit` defaults to `50` and must be between `1` and `1000`. Results are newest first. Each Job projection contains identifiers, kind/type, state, retry status, outcome, attempt limits, error kind, Patchset or Snapshot reference, availability, lock and update times, and tombstone state. Patchset CI jobs also include the current compact CI projection.

Read one exact Job:

```
curl --fail-with-body \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/worker-jobs/42"
```

Claim next or claim exact

Global claim applies an optional Repository filter:

```
{
  "accepted_job_kinds": [7, 11],
  "repository_indexes": [17],
  "accepted_runtime_contracts": ["ait.runner.native-job.v3"]
}
```

Send it to `POST /v1/native/worker-jobs:claim`. Both arrays must be duplicate free. An empty `repository_indexes` array means all serving Repositories. When nothing matches, the response is `200` with `claimed_job: null`.

To claim an already selected Job, POST this exact body to `.../worker-jobs/{worker_job_index}:claim`:

```
{"accepted_runtime_contracts":["ait.runner.native-job.v3"]}
```

A successful claim returns `attempt_count`, a hex `lease_token`, `lease_expires_at_s`, Job identity, and `runtime_request`. The runtime request selects a remote Snapshot, declares the direct `ci/run` argument vector, and sets a timeout. Execute that request exactly; do not substitute the current workspace.

Heartbeat, complete, and fail

Heartbeat requires the current attempt and lease and forbids `detail`:

```
{"attempt_count":1,"lease_token":"<hex-token>"}
```

POST it to `.../{worker_job_index}:heartbeat` before the lease expires.

Completion requires `detail`. For `repo.ci`, the fixed Job kind is sufficient for durable completion. For `patchset.ci`, `detail` must identify Job kind `7` and carry an `ait.runner.native-result.v1` result:

```
{
  "attempt_count": 1,
  "lease_token": "<hex-token>",
  "detail": {
    "job_kind": 7,
    "result": {
      "contract": "ait.runner.native-result.v1",
      "status": "succeeded",
      "tests_status": "pass",
      "suite_result_count": 1,
      "suite_results": [
        {"suite_id":"cargo_test","status":"pass","blocking":true}
      ]
    }
  }
}
```

Terminal result `status` is `succeeded`, `command_failed`, or `timed_out`; `tests_status` is `pass` or `fail`; each suite status is `pass` or `fail`. `suite_result_count` must equal the array length. The server derives compact CI evidence and atomically attaches it to the selected Patchset.

Failure requires an error and retry decision:

```
{
  "attempt_count": 1,
  "lease_token": "<hex-token>",
  "detail": {
    "retryable": true,
    "error": "runner lost its isolated attempt process"
  }
}
```

The `error` value is limited to 4,096 characters. The server requeues only when `retryable` is true and attempts remain. Complete, fail, and heartbeat reject a stale attempt count or lease token.

Lines and Snapshots

Read and update Lines

`GET .../lines` returns all Line projections. `GET .../lines/{line_name}` returns one projection including its name, status, and current `head_snapshot_id`.

Create or compare-and-set a Line with:

```
{
  "head_snapshot_id": "SNP-NEW",
  "expected_head_snapshot_id": "SNP-OLD"
}
```

Both fields are optional and nullable. `expected_head_snapshot_id` is the compare-and-set guard. Direct remote sync may bootstrap an empty default Line, replay its current head, or move another Line. It may not move an initialized default `main` to a different Snapshot; that returns `409` with `GOVERNED_TARGET_LINE_REQUIRES_LAND` and must be completed through Land.

Close a Line by POSTing to `.../lines/{line_name}:close`:

```
{"status":"archived"}
```

`status` defaults to `archived` when omitted.

Test Snapshot presence

```
curl --fail-with-body \
  -H 'Content-Type: application/json' \
  -d '{"snapshot_ids":["SNP-A","SNP-B"]}' \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/snapshots:exists"
```

The response classifies the supplied Snapshot IDs without transferring their content.

Export one Snapshot

```
curl --get --fail-with-body \
  --data 'include_content=true' \
  --data-urlencode 'path=src/main.rs' \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/snapshots/SNP-EXAMPLE"
```

`include_content` defaults to `true`. `path` optionally narrows the exported file projection. This is a JSON Snapshot export, not a raw pack download.

zstd bulk remote sync

The safe transfer order is: request a plan, upload missing raw packs, commit their manifest rows and Snapshots, then read or update Lines according to the governed-Line rules. Pull reverses the process by requesting a manifest and downloading its packs.

Presence plan

```
{
  "snapshot_ids": ["SNP-HEAD"],
  "object_packs": ["OPK-A", {"pack_id":"OPK-B"}],
  "tree_packs": ["TPK-A"]
}
```

POST to `.../remote-sync/zstd-bulk/plan`. The three arrays may be empty. Pack entries may be IDs or manifest objects carrying `pack_id`. The response preserves request order in:

- `checked_snapshot_ids`;
- `present_snapshot_ids` and `missing_snapshot_ids`;
- `present_object_pack_ids` and `missing_object_pack_ids`; and
- `present_tree_pack_ids` and `missing_tree_pack_ids`.

Raw pack upload and download

```
curl --fail-with-body -X PUT \
  -H 'Content-Type: application/vnd.ait.remote-sync.object-pack+zstd' \
  --data-binary @object-pack.zst \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/remote-sync/zstd-bulk/object-packs/O
    PK-A"
```

Tree Pack transfer uses the corresponding `tree-packs` path and media type. The path `pack_id` must be a valid single segment and must agree with the uploaded pack metadata. Upload responses describe the accepted pack; GET returns the exact raw bytes.

Commit

```
{
  "contract": "ait.remote_sync.zstd_bulk.commit.v1",
  "object_packs": [],
  "tree_packs": [],
  "blob_locators": [],
  "tree_locators": [],
  "snapshots": [],
  "line_update": null
}
```

Every inventory field accepts an array or `null`; missing fields are treated as empty. The pack, locator, and Snapshot objects are the exact rows emitted by an import or pull manifest. Do not synthesize or drop their integrity fields. `line_update`, when present, has `line_name`, nullable `head_snapshot_id`, and nullable `expected_head_snapshot_id`.

The response reports upserted and skipped Object Packs, Tree Packs, Blobs, Trees, and Snapshots, plus `remote_line`, `line_head_updated_after_ingest`, and `raw_binary_upload: true`. Existing immutable content is skipped only when it agrees with authority; a mismatch is an error rather than an overwrite.

Import and pull manifests

`GET .../import-manifests/{snapshot_id}` returns one complete closure:

```
{
  "contract": "ait.remote_sync.zstd_bulk.import_manifest.v1",
  "repo_name": "ait-native-site",
  "snapshot_id": "SNP-HEAD",
  "snapshots": [],
  "object_packs": [],
  "tree_packs": [],
  "blob_locators": [],
  "tree_locators": [],
  "line_update": null
}
```

For bounded ancestry, POST:

```
{
  "contract": "ait.remote_sync.zstd_bulk.pull_manifest.request.v1",
  "head_snapshot_id": "SNP-HEAD",
  "have_snapshot_ids": ["SNP-BOUNDARY"]
}
```

`have_snapshot_ids` must be duplicate free and is limited to 100,000 entries. The response contract is `ait.remote_sync.zstd_bulk.pull_manifest.v1` and adds `head_snapshot_id` plus `boundary_snapshot_ids`. Snapshot rows are ordered from the required ancestry boundary toward the requested head; pack inventories are deduplicated and dependency ordered. Download the returned pack IDs and pass the returned row arrays unchanged to the destination commit.

Task lifecycle

Create and read Tasks

Create an unplanned Task:

```
{"title":"Correct release readback","intent":"Verify the published artifact"}
```

Optional `task_id` is accepted only when it equals the next server-derived ID. Optional Plan linkage uses all three fields together:

```
{
  "title": "Correct release readback",
  "intent": "Verify the published artifact",
  "plan_id": "PR-12",
  "origin_plan_revision_id": "plan-revision:19",
  "plan_item_ref": "site/release/readback"
}
```

If `plan_id` is supplied without a revision, the route resolves the current head. If a revision is supplied without a Plan ID, it searches the selected Repository. The item must exist in that revision. Route-owned fields such as `repo_id` and `repository_index` are rejected.

Task responses contain `task_id`, Repository identity, sequence, title, intent, Plan linkage and drift projection, status, and timestamps. Status is `active`, `completed`, or `abandoned`.

List with `GET .../tasks`, read with `GET .../tasks/{task_ref}`, and close by POSTing to `.../tasks/{task_ref}:close`:

```
{"status":"completed"}
```

Accepted close status is `completed`, `abandoned`, `canceled`, or `cancelled`; the latter three project as `abandoned`.

Atomic Task start

`POST .../task-start` creates or selects the exact Plan revision and item, then appends the Task and first Change in one operation:

```
{
  "contract": "task-start-atomic/v1",
  "idempotency_key": "client-unique-key",
  "plan_item_ref": "site/release/readback",
  "plan": {
    "action": "existing",
    "plan_id": "PR-12",
    "plan_revision_id": "plan-revision:19"
  },
  "task": {
    "title": "Correct release readback",
    "intent": "Verify the published artifact"
  },
  "change": {
    "title": "Implement verified readback",
    "base_line": "main"
  }
}
```

`idempotency_key` is required and limited to 256 bytes. `plan.action` is:

- `existing`: requires `plan_id` and current-head `plan_revision_id`;
- `create`: requires a Plan revision payload under `plan.payload`; or
- `revise`: requires `plan_id`, `plan.payload`, and the current `expected_head_revision_id` unless the exact revision already exists.

The server derives Task Plan linkage and Change `task_id`; callers must not supply those derived fields. A replay with the same binding and payload returns the recorded atomic result.

Task audit and queues

`GET .../read/tasks/{task_id}/audit?target_line=main` returns the Task, all its Changes, target-Line head, counts of open and landed Changes, and a verdict: `task_completed`, `task_abandoned`, `ready_to_close`, `no_changes`, or `in_progress`.

`GET .../read/queue-summary` and `GET .../read/task-queue` accept optional `status=<value>`. The first returns aggregate workflow counts; the second returns Task queue rows. `GET .../read/reviewer-inbox` returns the Repository's current review work. These are derived read models: use entity routes as the write authority.

Change, Patchset, review, policy, CI, and Land

Create and close a Change

```
{
  "task_id": "T-0001",
  "title": "Implement verified readback",
  "base_line": "main",
  "fork_snapshot_id": "SNP-BASE"
}
```

`task_id`, `title`, and `base_line` are required. `fork_snapshot_id` is optional but, when supplied, must equal the base Line head at creation. Optional `change_id` must equal the server-derived full or short identity.

Change responses include `change_id`, full `change_ref`, Task and Repository, title, base Line, fork Snapshot, status, current and selected Patchset, land target, and timestamps. Status can be `draft`, `active`, `review`, `landed`, `archived`, `superseded`, or `abandoned`.

Close with one of these statuses:

```
{"status":"archived"}
```

Accepted values are `archived`, `superseded`, `abandoned`, `canceled`, and `cancelled`. `landed` is rejected because successful Land derives that state.

Publish and select a Patchset

POST to `.../changes/{change_ref}/patchsets`:

```
{
  "base_snapshot_id": "SNP-BASE",
  "revision_snapshot_id": "SNP-REVISION",
  "summary": "Add verified readback",
  "author_mode": "ai_with_human_review"
}
```

Required author modes are `human_only`, `human_with_ai_assist`, `ai_with_human_review`, or `ai_only_experimental`. Optional `patchset_id` and `patchset_number` are accepted only when they match the next derived identity. Replaying the same Change/base/revision triple returns the existing Patchset.

Patchset responses include identity, Snapshots, summary, author and publish state, withdrawal flags, changed-file statistics and paths, evaluation state, creation time, CI run sequence, and compact overall/tests/lint evidence.

Select by POSTing to `.../changes/{change_ref}:selectPatchset`:

```
{"patchset_id":"T-0001/C-01/P-02"}
```

The Patchset must belong to the Change and must not be withdrawn or invalidated.

Request and record review

Request exactly one reviewer group:

```
{
  "patchset_id": "T-0001/C-01/P-02",
  "reviewer_groups": ["maintainers"],
  "note": "Please check release semantics"
}
```

Send it to `.../changes/{change_ref}:requestReview`. Optional `review_id` is accepted only when it matches the derived identity.

Record a review by POSTing to `.../changes/{change_ref}/reviews`:

```
{
  "patchset_id": "T-0001/C-01/P-02",
  "reviewer": "reviewer@example.invalid",
  "action": "approve",
  "comment": "Verified",
  "blocking": false
}
```

Actions are `request`, `comment`, `task_comment`, `code_review_summary`, `approve`, `task_approve`, `request_changes`, `task_request_changes`, `defer`, `task_defer`, and `dismiss`. `comment` defaults to empty and `blocking` to false. The Patchset must be the selected Patchset for its Change.

The review list response contains `reviews` plus approval, blocking, comment, and total summary counts.

Attestation and policy

Append an attestation with `PUT .../patchsets/{patchset_id}/attestation`:

```
{
  "author_mode": "ai_with_human_review",
  "verification_state": "pass",
  "revoked": false,
  "require_tests_pass": true,
  "require_human_review": true,
  "require_lint_pass": true
}
```

`verification_state` is `unknown`, `pending`, `pass`, or `fail`. Defaults are `unknown`, `revoked: false`, `require_tests_pass: true`, and both human-review and lint requirements false. If `author_mode` is supplied, it must agree with the Patchset. Caller-supplied `detail.patchset_ci` is rejected because CI completion belongs to the Patchset and Worker Job authority.

`GET .../attestation` returns the latest attestation or `404` when none exists. It includes requirement flags, whether CI backed the record, author mode, and the current tests/lint summary.

`POST .../patchsets/{patchset_id}:evaluatePolicy` accepts an empty JSON object and appends a decision based on the latest attestation, compact CI, and live review approvals. `GET .../policy` returns that decision and its checks. Before the first evaluation it returns:

```
{"patchset_id":"...","decision":"pending","checks":[]}
```

Run and inspect Patchset CI

Start CI with:

```
{"trigger":"manual_rerun"}
```

POST to `.../patchsets/{patchset_id}:runCi`. An absent trigger defaults to `manual_rerun`. `manual_rerun` and `base_stale_after_land_rerun` explicitly advance the run; another non-empty trigger reuses existing terminal evidence when available. A new or still-active run is delivered through a Worker Job. The response reports `queued`, `job`, `delivery`, `trigger`, and the Patchset CI projection.

Inspect with:

```
curl --get --fail-with-body \
  --data 'recent_limit=10' \
  --data 'projection=readiness' \
  "$BASE/v1/native/repository-authorities/$REPOSITORY_INDEX/read/patchsets/<patchset-id>/ci-stat
    us"
```

`recent_limit` defaults to `10` and must be between `1` and `1000`; the only named projection is `readiness`. The response reports availability, run sequence, completion time, status and counts, whether runnable evidence exists, the selected/latest Job, and recent Jobs. 1.1.1's compact authority does not retain full per-suite history here, so `selected_suite_ids` and `suite_results` are empty in this read model.

Atomic Task Land

Prefer this route over direct Change `:submit`:

```
{
  "contract": "task-land-atomic/v1",
  "idempotency_key": "client-unique-land-key",
  "task_or_change_ref": "T-0001/C-01",
  "target_line": "main",
  "mode": "direct",
  "patchset_id": "T-0001/C-01/P-02",
  "expected_head_snapshot_id": "SNP-BASE"
}
```

`contract`, `idempotency_key`, `task_or_change_ref`, and `mode` are required. The key is limited to 256 bytes. `task_or_change_ref` may be a Task only when that Task has exactly one landable Change; otherwise send the exact Change. `target_line` defaults to the Change base Line. Mode is `direct`, `merge`, or `ff-only`. `patchset_id` is optional but must be selected. The compare-and-set guard may be named `expected_head_snapshot_id` or `expected_target_line_head`.

Success or replay returns the atomic contract, replay flag, top-level Task, Change, Patchset, target and landed Snapshot identifiers, plus full `task`, `change`, `patchset`, `land`, and optional `history_promotion` projections.

Direct `POST .../changes/{change_ref}:submit` accepts the Land fields without the atomic contract requirement. It is lower level and does not provide the same Task resolution and idempotent aggregate result.

Read any attempt with `GET .../lands/{submission_id}`. A Land projection includes selected Patchset, target Line, mode, status, failure kind, result, landed Snapshot, Line action, and timestamps. Failure kinds include stale base, blocked policy/review/CI, conflict, target update failure, and internal error.

History promotion prepare

This specialized operation converts recorded local-land history into remote workflow records before governed remote Land. The top-level request is:

```
{
  "contract": "history-promotion-prepare/v1",
  "idempotency_key": "client-unique-promotion-key",
  "target_line": "main",
  "base_snapshot_id": "SNP-BASE",
  "revision_snapshot_id": "SNP-FINAL",
  "author_mode": "ai_with_human_review",
  "summary": "Promote recorded local history",
  "entries": []
}
```

`entries` must contain 1 through 64 items. Each item contains:

Field	Meaning
`local_task_id`, `local_change_id`, `local_change_ref`	Original local identities.
`expected_remote_task_id`, `expected_remote_change_ref`	Optional fail-closed publication identities.
`task`	`{title, intent, plan_id?, origin_plan_revision_id?, plan_item_ref?}`.
`change`	`{title, base_line, fork_snapshot_id}`.
`pre_land_target_snapshot_id`, `landed_snapshot_id`, `landed_at_s`	Recorded local Land boundary.
`snapshots`	One to 64 `{snapshot_id, created_at_s}` entries for that boundary.

The endpoint validates ordering, identity, Snapshot ancestry, Plan bindings, and idempotency before preparing the aggregate Patchset. Raw callers should use the exact receipt generated by the compatible `ait` client rather than constructing this payload by hand.

Sprint and Plan API

The HTTP path uses `sprints`; response objects use Plan identifiers. A Plan revision is backed by one Markdown artifact plus normalized item metadata.

Plan revision payload

Create and revise share these fields:

Field	Required	Meaning
`title`	create: yes; revise: no	Plan title or revision title override.
`status`	no	`draft`, `active`, `archived`, or `superseded`; default is `draft`. `active` is stored and projected as the draft/open state.
`summary`	no	Revision summary.
`artifact_path`	yes	Repository-relative Markdown artifact path.
`artifact_selector`	no	Stable selector within the artifact.
`artifact_heading`	yes	Human-readable root heading.

Field	Required	Meaning
`items`	no	Normalized item array; defaults empty.
`artifact_body`	no	Exact Markdown body used to pack the revision artifact.
`packed_artifact`	no	Existing packed artifact descriptor when used by a compatible client.
`expected_head_revision_id`	revise only, no	Compare-and-set guard for concurrent revision.

Each `items[]` object may contain `plan_item_ref`, item text, `checkbox_state`, and `heading_path`. Items without a usable reference cannot be selected by Task linkage.

Create with `POST .../sprints`. Caller-supplied `plan_id` is rejected because the server derives it. List with `GET .../sprints`; optional `artifact_path=<exact-path>` narrows the result. Read one with `GET .../sprints/{plan_id}`.

Update status with:

```
{"status":"archived"}
```

Append a revision by POSTing the shared revision payload to `.../sprints/{plan_id}/revisions`. List or read revisions with the matching GET routes.

The `/artifacts` GET route returns the same full revision projection as the revision GET route. The corresponding PUT is present for protocol compatibility but 1.1.1 always rejects it because separate supporting-artifact writes are not part of the active compact Plan layout. Put the artifact body in the Plan create or revise operation instead.

Plan lookup and Task linkage helpers

Find active Plans containing all supplied normalized terms:

```
{"contains_terms":["release","readback"]}
```

POST to `.../sprint-plan-ids/by-contains`. Terms must be a JSON string array; an empty normalized set returns an empty array.

Resolve Task linkage by POSTing any combination of:

```
{
  "plan_id": "PR-12",
  "origin_plan_revision_id": "plan-revision:19",
  "plan_item_ref": "site/release/readback"
}
```

No Plan and no revision returns all three fields as `null`. An item without Plan linkage is rejected. Missing Plan or revision identity is resolved within the numeric Repository, then ownership and item existence are validated.

`GET .../read/plans/candidate-inputs?contains=release,readback` normalizes the comma-separated terms and returns matching active Plans plus linked Task candidates. It is a read model for selection UI, not a mutation endpoint.

End-to-end raw HTTP example

This abbreviated sequence shows how the lower-level APIs connect. A production client must preserve every returned identifier and stop on any non-success status.

1. `GET /healthz` until `200` and `ready: true`.
2. `GET /v1/handshake`; verify protocol and runner contracts.
3. Register once with `POST /v1/native/repository-authorities`; persist the numeric Repository index.
4. Use remote-sync plan, raw pack upload, and commit to publish immutable Snapshot content without advancing an initialized governed `main` Line.

5. Create or atomically start the Plan-bound Task and Change.
6. Publish and select the Patchset whose revision Snapshot was uploaded.
7. Add attestation and any required review request.
8. POST Patchset `:runCi`; an `ait-runner` claims, heartbeats, and completes the resulting Worker Job.
9. POST Patchset `:evaluatePolicy`; inspect the returned decision.
10. POST `/task-land` with an idempotency key and expected target head.
11. Read the Land and Task audit; only a successful Land advances governed `main` and permits final Task completion.

For ordinary use, let `ait workflow ready`, `ait-runner`, and `ait task finish` perform this sequence. The direct API exists for compatible integrations and operators who need explicit transport-level control.

Operator verification checklist

- The listener is private or protected by a trusted TLS ingress.
- Health and handshake are checked separately; the compact endpoint list is not treated as the full API catalog.
- Every request is routed by the saved numeric Repository index.
- Mutating retries use operation idempotency keys where the contract provides them and compare-and-set heads where Lines can move.
- `503` respects `Retry-After`; `409` is investigated rather than blindly retried.
- Binary media types, byte limits, pack IDs, file sizes, and digests are validated exactly.
- Retirement export is complete and independently durable before purge.
- Restore is rehearsed on an isolated server and the newly assigned Repository index is recorded.
- Worker lease tokens stay in memory or protected transient state and are never used as general API authentication.

See `ait-server: Remote Authority and Recovery` for deployment, preservation boundaries, and disaster-recovery procedure, and `Remote Infrastructure` for the server/runner operating boundary.

INFRASTRUCTURE AND INTEGRATIONS

ait-agent: Optional Transport Manager

Understand the optional 1.1.1 transport-manager boundary, its four worker families, lifecycle commands, and relationship to the coding client and native worker executable.

AUDIENCE Agent operators, integration maintainers, and Repository owners

https://ait-native.dev/technical/v1.1.1/ait-agent/

Role in ait-native

`ait-agent` is the optional native manager for Repository-scoped Telegram, LINE, Discord, and Slack workers. It is a standalone executable, separate from the main `ait` CLI: 1.1.1 has no root `ait agent` namespace, so an operator installs and supervises the transport surface through `ait-agent` directly.

It is not the primary coding application. Normal interactive work continues in the coding client that already owns the user conversation and Repository session. Install this component only when a headless transport endpoint is a real operating requirement.

Manager and worker boundary

Component	Responsibility
`ait-agent`	Add, list, inspect, start, stop, restart, read logs, and remove named worker configurations. Telegram also exposes a supervisor lifecycle.
`ait-agent-worker`	Run the selected native transport, verify ingress, invoke the bounded reply provider, and deliver the reply.
Coding client	Own the main interactive coding experience and decides when authorized AIT workflow commands are run.
`ait-server`	Own optional remote Repository authority, offsite preservation, recovery state, and Worker Jobs.

Starting a transport worker does not itself create a sprint card, Task, Snapshot, Patchset, or Land. Those remain explicit repository workflow operations performed by an authorized client.

Public command families

All four transports expose these lifecycle groups:

```
add  list  status  start  stop  restart  logs  remove
```

Telegram additionally exposes:

```
supervisor status  supervisor start  supervisor run
supervisor stop    supervisor restart
```

Read-only orientation examples are:

```
ait-agent telegram list --json
ait-agent telegram status main --json
ait-agent telegram logs main --lines 100
ait-agent telegram supervisor status --json
```

The add commands accept transport credentials and lifecycle paths. Their exact required options differ by transport; use the generated standalone `ait-agent` reference instead of copying options between Telegram, LINE, Discord, and Slack.

Configuration and secret boundary

Manager commands write named entries in `.ait/agent-workers.json` and maintain their runtime metadata. Credentials must be handled as secrets: avoid retained shell history when a command requires a token, restrict the worker environment and runtime files, and never publish logs or configuration copies without redaction.

The field-by-field manifest, credential precedence, transport defaults, reply provider, and sandbox controls are documented in Agent-worker Configuration. The executing process contract and signed one-shot commands are documented in ait-agent-worker.

Operating decision

Use `ait-agent` when a managed external transport materially helps an operator reach one Repository. Do not add it merely to reproduce a coding-client UI or to place messaging inside the core architecture. The local CLI, native bindings, Task isolation, Binary DB, and optional server/runner path remain complete without this manager.

INFRASTRUCTURE AND INTEGRATIONS

ait-agent-worker: Messaging and Reply Runtime

Operate the optional native messaging worker through its verified command surface, four transports, Codex reply provider, runtime admission, and security boundary.

AUDIENCE Agent operators, integration maintainers, and Repository owners

https://ait-native.dev/technical/v1.1.1/ait-agent-worker/

Role in ait-native

`ait-agent` and `ait-agent-worker` are separate release-family components from the same `ait-core` source authority. `ait-agent` manages worker configuration and processes; `ait-agent-worker` is the Python-free native transport and reply-execution runtime for Telegram, LINE, Discord, and Slack.

The worker is not required for ordinary `ait` CLI use and it is not a separate desktop or mobile coding client. A primary coding client remains responsible for the interactive authoring experience. `ait-agent-worker` accepts a message from one explicitly configured transport, binds it to one Repository context, obtains a bounded reply, and returns that reply through the same transport.

The worker does not automatically create a sprint card, start a Task, create a Snapshot, or Land a Change. Those workflow mutations occur only when the reply provider or another authorized client explicitly runs the corresponding `ait` commands.

Appropriate uses

- Operate one named messaging bot against one AIT Repository without a Python worker runtime.
- Receive Telegram, LINE, Discord, or Slack requests through the transport mode implemented for that service and return Repository-contextual replies.
- Place a native, headless worker behind a trusted service supervisor or HTTP adapter.
- Inspect the exact transport, event-loop, platform, and fallback capabilities compiled into an installed executable before admitting it to service.
- Use the worker's native Codex reply provider, or supply one explicit custom local reply program for a controlled integration.

Use `ait-server` rather than `ait-agent-worker` for remote Repository authority, offsite preservation, recovery, Patchset CI, and runner job coordination. The worker may resolve a local or remote Repository target, but it is not the server's administration API.

Message-to-reply execution path

1. The process resolves the Repository root, `.ait/config.json`, and the worker manifest. `AIT_AGENT_CONFIG_PATH` may select a different manifest path.
2. The manifest is normalized and the exact `<kind>/<name>` worker is selected. Invalid fields, a missing worker, or a kind/name mismatch fail before the transport starts.
3. Credentials and runtime settings are resolved. The effective workflow mode determines whether the Repository target is local or uses its configured default remote.
4. The requested native event-loop backend and zero-based shard are checked against the runtime admission plan. A backend or shard mismatch fails closed.
5. The transport authenticates or verifies its service-specific ingress, parses the exact request, applies its conversation and duplicate state, and submits a bounded reply job.
6. The configured local reply program runs. When no complete custom provider is supplied, the native worker installs its own `reply-provider` subcommand and invokes Codex with the selected model, reasoning effort, sandbox, and

timeout.

7. The transport sends or returns the reply. Failures use a structured native diagnostic and do not fall back to a Python worker.

The worker's sync state and Codex thread binding keep transport conversation identity separate from AIT workflow authority. A resumed conversation can reuse its compatible Codex thread, while Task, Change, Snapshot, and Land state remain owned by AIT.

Complete executable command surface

Command	Function, input, and result
`capabilities`	Reports the installed platform, architecture, socket and process-control backends, supported transports, event-loop backends, default backend, and Python-fallback state. Text is the default; `--json` emits `ait.agent.worker.capabilities.v1`.
`run`	Runs one exact named worker after configuration normalization and runtime admission. It requires transport, worker name, event-loop backend, and shard. Service mode is long lived. Telegram also admits one stdin webhook transaction through console mode.
`slack-command`	Executes one signed Slack slash-command transaction. It reads the exact form body from stdin, verifies request metadata supplied by the trusted HTTP boundary, and writes a redacted JSON outcome.
`discord-interaction`	Executes one signed Discord interaction transaction. It reads the exact JSON body from stdin, verifies request metadata supplied by the trusted HTTP boundary, and writes the Discord response object.
`reply-provider`	Executes one versioned native Codex reply-provider request. It reads JSON from stdin and writes `ait.agent.gateway_reply_provider_response.v1` JSON.

Exact 1.1.1 forms:

```
ait-agent-worker capabilities [--json]

ait-agent-worker run \
  --transport <telegram|discord|slack|line> \
  --worker <name> \
  --event-loop-backend <backend-reported-by-capabilities> \
  --shard <zero-based-index> \
  [--console-mode <service|webhook>]

ait-agent-worker slack-command [--worker <name>] \
  --signature <signature> --signature-timestamp <unix-seconds> < exact-slack-body
ait-agent-worker discord-interaction [--worker <name>] \
  --signature <signature> --signature-timestamp <timestamp> < exact-discord-body
ait-agent-worker reply-provider < provider-request.json
```

`service` is the default `run` console mode. Use `webhook` only for one Telegram webhook payload supplied on stdin. The Slack and Discord one-shot commands are adapter boundaries, not unauthenticated network listeners: the caller must preserve the exact raw body and supply the matching signature and timestamp metadata through the trusted request boundary.

Before launch, record both forms of capability evidence:

```
ait-agent-worker --version
ait-agent-worker capabilities
ait-agent-worker capabilities --json
```

Do not infer a backend or transport from the operating-system name alone. Use the installed binary's capability result, then pass the backend and shard selected by launch planning to `run`.

Transport behavior

Transport	Long-lived service	Bounded request mode	Authentication and delivery boundary
Telegram	Polling or configured webhook service.	`run --console-mode webhook` consumes one webhook payload from stdin.	Bot token is mandatory. Webhook mode can require its configured webhook secret. Replies can use message merging, Markdown, decoupled delivery, and optional local speech-to-text settings.
LINE	HTTP callback service on the configured host, port, and path.	No separate public one-shot subcommand.	Channel secret verifies callback signatures; the channel access token owns reply delivery through the configured LINE API base URL.
Discord	Gateway service when the selected mode has the required bot credential.	`discord-interaction` handles one signed interaction body.	Application identity selects the worker. Public-key verification and bot delivery credentials are required by the selected interaction or gateway mode.
Slack	Socket Mode when the selected worker has its app token.	`slack-command` handles one signed slash-command form body.	The signing secret verifies HTTP commands. Socket and HTTP modes use their own required credentials; deferred command delivery is bounded by the command transaction.

All listener defaults are loopback addresses. Put an HTTP listener behind a trusted reverse proxy only after request verification, TLS, body-size, timeout, and forwarding behavior have been tested with the exact selected transport.

Native Codex reply provider

Transport workers need a reply provider; they do not contain a second coding model. An explicit complete `local_reply.program` plus `local_reply.args` configuration wins. Otherwise the native executable configures itself as the provider by running its `reply-provider` subcommand.

The native provider accepts the versioned `ait.agent.gateway_reply_provider_request.v1` JSON contract. The request binds one Repository root, conversation key, surface, actor, input payload, optional prior provider thread, and reply settings. It starts or resumes a Codex thread for that compatible Repository and conversation, captures the final reply and bounded turn telemetry, and returns the versioned response contract.

The provider enforces bounded request and process-output sizes, a thread lock, the configured timeout, and one of `read-only`, `workspace-write`, or `danger-full-access` sandbox values. Sandbox selection is the execution boundary. It must match what the worker is allowed to do in that Repository; transport credentials do not grant additional filesystem or AIT authority.

Admission, concurrency, and shutdown

`run` never chooses an arbitrary shard. It normalizes the full worker manifest, plans the configured worker set against the selected native event-loop backend, and verifies that the requested worker belongs to the supplied shard. Telegram can additionally bound expected concurrent workers and workers per shard.

Transport reply work uses bounded native job admission. When capacity is full, the worker reports failure instead of creating unbounded background work. A service stops accepting new jobs before draining admitted work; process-control behavior is the native backend reported by `capabilities`.

Failure and security contract

Operational failures are written to stderr as `ait.agent.worker.error.v1` JSON. The diagnostic contains a stable code, message, numeric exit code, bounded details, and `python_worker_execution_allowed: false`.

Exit code	Meaning
2	Invalid request, including malformed input or rejected signed ingress.
3	Invalid worker, manifest, credential, backend, or shard configuration.
4	Required native runtime, transport, process, or output boundary unavailable.

Diagnostics report credential presence, not values. Slack and Discord verify the exact raw body. One-shot outcomes redact secrets and delivery locators. Python fallback is disabled for 1.1.1 transport, signature, reply, and state work.

Deliberate limits

- `ait-agent-worker` is not a general-purpose desktop, mobile, or web app.
- It is not the durable remote authority and does not replace `ait-server`.
- Receiving a message does not itself create or complete AIT workflow objects.
- A transport worker is scoped to one resolved Repository and one named worker selection per process invocation.
- The public commands do not bypass transport signatures, manifest validation, runtime admission, Codex sandboxing, or AIT workflow policy.

Configuration map

- Appendix: Agent-worker Configuration expands `.ait/agent-workers.json`, credential precedence, every transport field, local reply settings, runtime paths, and verification guidance.
- Appendix: Environment Variables identifies the supported worker bootstrap and credential environment inputs.
- ait-server: Remote Authority and Recovery explains the separate remote authority, preservation, and recovery boundary.

INFRASTRUCTURE AND INTEGRATIONS

Python Integration

Use the direct in-process Python binding included in the ait-native PyPI distribution.

AUDIENCE Python integrators

https://ait-native.dev/technical/v1.1.1/integrations/python/

Package boundary

The PyPI project is `ait-native`. Its platform wheel includes the admitted native commands and the direct in-process Python binding from the same 1.1.1 compatibility family.

```
python -m pip install ait-native==1.1.1
```

The binding does not download a runtime, launch an `ait` subprocess as its API transport, or provide a separate workflow implementation.

Direct API

```
from ait_python import AgentClient, NativeRuntime

runtime = NativeRuntime()
print(runtime.binding_info())

agent = AgentClient(runtime)
print(agent.capabilities().supported_transports)
```

`NativeRuntime` exposes versioned native exports in process. `AgentClient` provides typed access to supported agent capabilities and worker operations.

Repository workflow

Installing the Python package does not initialize a repository. Run `ait init` once from the target repository, then let the coding agent follow the generated instructions.

```
ait init
ait status --json
```

Python projects do not receive a different AIT workflow. Test, lint, build, and packaging commands remain repository-authored inputs.

Version check

Read binding metadata before combining it with another component. An independently rebuilt binding or mismatched core is not an admitted 1.1.1 family merely because its Python package imports.

See Distribution and Release for the family and license boundaries.

INFRASTRUCTURE AND INTEGRATIONS

Node.js Integration

Use the direct Node-API JavaScript and TypeScript surface from the supported npm package.

AUDIENCE Node.js integrators

https://ait-native.dev/technical/v1.1.1/integrations/node/

Package boundary

The supported npm identity is `@wa120/ait-native`. It contains the portable JavaScript and TypeScript facade and selects the exact 1.1.1 platform Node-API addon.

```
npm install @wa120/ait-native@1.1.1
```

The package does not bundle or start `ait-server`. Its API loads the package-owned addon directly and does not use a child process as the binding transport.

Direct API

```
import { NativeRuntime } from "@wa120/ait-native";

const ait = new NativeRuntime();
console.log(ait.bindingInfo());
const status = ait.runCli(["status", "--json"]);
console.log(status);
```

The packaged `ait` command calls the same native binding in process. The JavaScript API and command therefore share the admitted Rust behavior.

Repository workflow

Node.js repositories use the same initialization and generated repository instructions as every other supported repository.

```
npx --no-install ait init
npx --no-install ait status --json
```

AIT does not inspect `package.json` to choose test commands or framework behavior. The repository remains responsible for its explicit validation.

Platform packages

Target-specific addon packages are exact-version implementation dependencies, not separate products. Applications should depend on the supported top-level package and let normal npm resolution select the matching addon.

BINARY DB V0 SCHEMA

Binary DB v0: Format and Authority

Read the Binary DB v0 file format, integer and offset rules, and storage-root boundaries.

AUDIENCE Core implementers, integrators, and operators

https://ait-native.dev/technical/v1.1.1/binary-db/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

This chapter is the entry point to the complete active Binary DB v0 technical schema. It explains how every fixed record, payload, and index is addressed and which storage root owns it.

Unassigned behavior remains reserved and fails closed: an implementation must not invent an extended-handle encoding, silently widen a field, or select a new canonical payload encoding outside a declared layout conversion.

Global File Format

Every persisted Binary DB `.bin` file and every rebuildable `.idx` file starts with one four-byte header:

```
BIN_HEADER_SIZE = 4
INDEX_HEADER_SIZE = 4

BinHeader / IndexHeader:
u32 layout_id = 1     # little-endian
```

The file path determines record kind, namespace, authority scope, and whether the file is authoritative storage, payload storage, object data, or an optional cache. The header contains no magic, record count, checksum, authority name, generation token, or duplicate file kind.

For fixed-record files:

```
record_count  = (file_size - BIN_HEADER_SIZE) / record_size
record_offset = BIN_HEADER_SIZE + record_index * record_size
```

`file_size` must be at least four bytes and the record area must be exactly divisible by the selected record size. One file must never mix records from different layouts. Record codecs use the declared byte offsets and do not rely on Rust struct layout, host alignment, or implicit padding.

Payload offsets are `u64` file addresses measured from byte zero of the owning payload file. The first payload normally begins at offset `BIN_HEADER_SIZE`. Payload and fixed-record integer fields use the repository's fixed byte order. Reserved bits and fields are written as zero.

Unless a field is the separately declared signed Git identity timestamp, every active fixed-record `*_at_s` field is a little-endian unsigned `u64` count of whole seconds since the Unix epoch. Pre-epoch input fails closed. Zero retains the exact absent or unknown meaning declared for that field. No ordering or uniqueness is inferred from timestamp precision.

`*_index_plus1 = 0` means absent; non-zero values encode `index + 1` because record index zero is valid. An index field without the `plus1` suffix stores a direct record index and may legitimately contain zero.

Authority Scopes

The same filename may use a different record layout when its authority root is different:

- Local workflow authority owns local Task, Change, Line, Tag, Stash, Land, Land-target-Line, task-snapshot-link, and Plan state.
- One `ait-server` repository authority owns remote Task, Change, Line, Land, Land-target-Line, task-snapshot-link, Plan, Patchset, Worker Job and its ready/state indexes, Attestation, Actor, Review, Policy, and Waiver state.
- One `ait-server` installation owns a distinct server-global operational authority root only for the Repository registry and its namespace lookup index. A `repository_index` points to that root's `repository.bin` and routes to one server Repository authority. A Worker Job identity is meaningful only as `(repository_index, worker_job_index)`; the second component is the permanent physical record index local to the routed Repository authority.

- Shared content authority owns content Snapshot, ordered Snapshot-parent, Tree, normalized Tree-entry-range, Blob, and object-pack metadata.
- Optional local Git interoperability authority owns Git import/export repository identities, generations, immutable mappings, and resumable checkpoints. It is an escape-hatch mapping authority, not primary AIT workflow or content authority.
- A task worktree owns disposable `.ait-worktree/cursor.bin` state.
- Optional local content caches own manifest/file lookup acceleration.
- An optional local remote-mirror root may mirror server snapshot links while preserving server indexes.

Equal numeric indexes in different authority roots are not the same identity. Local writers never allocate server indexes.

BINARY DB V0 SCHEMA

Binary DB v0: Workflow Records

Expand Task, Change, worktree, Line, Stash, Tag, Land, and inline workflow-link record layouts and invariants.

AUDIENCE Core implementers and workflow integrators

https://ait-native.dev/technical/v1.1.1/binary-db/workflow/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

These layouts are the fixed and linked records for local and remote workflow state. Read each byte layout together with the following identity, mutation, validation, and legacy-conversion rules; the record-size constant alone is not the complete contract.

Task Records

```
LOCAL_TASK_RECORD_SIZE = 64

LocalTaskRecord - task.bin:
u8  task_meta
u8  local_meta
u16 payload_len
u64 payload_offset
u32 origin_plan_revision_index_plus1
u32 plan_item_index_plus1
u32 published_remote_task_index
u64 created_at_s
u64 updated_at_s
u64 plan_linked_at_s
u64 published_at_s
u64 closed_at_s
```

```
REMOTE_TASK_RECORD_SIZE = 60

RemoteTaskRecord - task.bin:
u8  task_meta
u8  remote_meta
u16 payload_len
u64 payload_offset
u32 origin_plan_revision_index_plus1
u32 plan_item_index_plus1
u64 created_at_s
u64 updated_at_s
u64 plan_linked_at_s
u64 fetched_at_s
u64 closed_at_s
```

Task identity is derived rather than stored:

```
task_index = record_number
task_seq   = task_index + 1
task_id    = derive(authority_scope, repo_namespace, "T", task_seq)
```

Task records are append-only and must never be physically reordered. A legacy SQLite import may compact sparse legacy sequences once; it must not preserve holes by inserting padding records.

`closed_at_s` is a fixed tail field of `LocalTaskRecord` and `RemoteTaskRecord`, not payload and not a separate record or file. Zero means either that the effective Task state is not terminal or that an offline legacy conversion had no source close time for a terminal Task. In the latter case the Task is closed and its historical close time is unknown. A non-zero value is the close time and is valid if and only if the effective Task state is terminal under `TaskRecord.task_meta` and, for local authority, `LocalTaskRecord.local_meta`. Later publication or linkage updates may change `updated_at_s` but never replace the close time.

Task creation writes `closed_at_s = 0`. Close writes and fsyncs `closed_at_s` before writing and fsyncing the terminal status bits as the commit marker. Native writers never clear terminal status to reopen a Task. Readers treat a stale non-zero

time on a non-terminal Task as absent, and recovery repairs it to zero. Native v0 writers must never create a terminal Task with zero close time. Because no migration-provenance bit or field exists, readers, validators, and recovery accept terminal plus zero as an unknown historical close time and do not classify that representation as corruption.

The pre-correction layout-1 local 40-byte and remote 36-byte Task records are offline conversion input only. A converter must receive the source record size explicitly and must never infer it solely from file-size divisibility. Under exclusive authority lock it rewrites the complete `task.bin` into a separate file using the 44-byte or 40-byte record and preserves every close time supplied by source authority. For a terminal Task whose legacy source format demonstrably did not record close time, it writes `closed_at_s = 0` and must not substitute creation time, update time, conversion time, or any other inferred timestamp. A source that defines close time but supplies an invalid value still fails closed. The converter validates the entire rewritten file and atomically replaces the old file before activation. One active `task.bin` never mixes the pre-correction and corrected record sizes.

For bin-to-bin conversion, source planning states `unplanned` and `explicit_unplanned` both encode as `TaskRecord.task_meta` bit 0 clear; `planned` encodes as bit 0 set. This is the complete v0 planning-state encoding: v0 intentionally does not retain a third distinction between the two unplanned source spellings. Numeric Plan revision and item bindings are resolved and preserved independently. A textual `plan_section_ref` or `plan_drift_state` is accepted without another field only when it is absent or exactly derivable: section ref comes from the resolved Plan Item's `heading_path_bytes`, while drift state comes from the numeric binding and the current Plan/Plan-revision metadata. A non-derived or ambiguous value fails closed.

An admitted source Task status `abandoned` maps to `TaskRecord.task_meta` bit 7 `canceled`. It does not create another Task state field and does not infer `LocalTaskRecord.local_meta` bit 1 from a server source. Under the separately enumerated exact locked-source gate, the legacy source spelling `canceled` maps to that same existing bit and no other state. A local source explicitly carrying the independent local-abandonment state preserves that local bit under its existing semantics.

For an offline conversion from a legacy Remote Task format that demonstrably has no update or fetch time, the converter writes the corresponding `updated_at_s = 0` or `fetched_at_s = 0`. Each zero means unknown historical time, not the conversion time. A supplied valid time is preserved exactly; a source format that defines either field but supplies an invalid value fails closed. Native v0 Remote Task writers continue to write their actual event times and do not use this legacy exception.

Change Records

```
TASK_CHANGE_INDEX_RECORD_SIZE = 8

TaskChangeIndexRecord - task_change_index.bin:
u32 latest_change_index_plus1
u16 change_count
u8  next_change_ordinal
u8  reserved0
```

```
LOCAL_CHANGE_RECORD_SIZE = 68

LocalChangeRecord - change.bin:
u8  change_meta
u8  local_meta
u16 payload_len
u8  change_ordinal
u8  change_state
u16 reserved1
u64 payload_offset
u32 task_index
u32 previous_change_index_plus1
u32 fork_snapshot_index_plus1
u8  published_remote_change_ordinal_plus1
u8  reserved2
u16 reserved3
u64 created_at_s
u64 updated_at_s
u64 published_at_s
u32 base_line_index_plus1
u64 archived_at_s
```

```
REMOTE_CHANGE_RECORD_SIZE = 68

RemoteChangeRecord - change.bin:
u8  change_meta
u8  remote_meta
u16 payload_len
u8  change_ordinal
u8  change_state
u16 reserved1
u64 payload_offset
u32 task_index
u32 previous_change_index_plus1
u32 selected_patchset_index_plus1
u32 fork_snapshot_index_plus1
u64 created_at_s
u64 updated_at_s
u64 fetched_at_s
u32 base_line_index_plus1
u64 archived_at_s
```

The Change record stores neither textual Change ID nor textual Task ID. Its identity is `(authority_scope, task_index, change_ordinal)`.

A published Local Change's Remote identity is the pair formed by its owning Local Task's `published_remote_task_index` and `published_remote_change_ordinal_plus1 - 1`. The Task field selects the exact Remote Task ordinal; the Change field selects `C-01..C-64` inside that Task. No Local Change stores or requires a global Remote `change_index`.

For a Remote Change, `ChangePatchsetIndexRecord.latest_patchset_index_plus1` is the latest/current Patchset and `selected_patchset_index_plus1` is the sole persisted mutable selected-Patchset authority used by review, readiness, and Land. Selection is Change state, not an immutable Patchset property. Zero means no selected Patchset. A non-zero selected pointer must reference a live Patchset owned by this Change. Source `current_patchset_number` resolves through the latest index; source `selected_patchset_number` resolves through the selected pointer. They may differ and must never be collapsed into one value. Selection updates this existing pointer without mutating a Patchset record. This is a semantic rename of the existing four-byte slot and does not shift that slot or any field in the 44-byte prefix. The later inline lifecycle extension appends only the two declared tail fields.

For the admitted legacy server format only, a landed Change that has one or more surviving Patchsets may supply `current_patchset_number = 0` even though latest/current remains rebuildable. Conversion accepts that zero only when the source selected pointer is non-zero and resolves to the surviving Patchset owned by the Change with the greatest `patch_ordinal`; it writes that Patchset as `ChangePatchsetIndexRecord.latest_patchset_index_plus1`. Any other zero, missing, ambiguous, non-owned, omitted, or disagreeing latest evidence fails closed. The selected pointer remains independently stored and is not inferred from this exception.

`ChangeRecord.change_state` bit 0 is `canceled`; bits 1 through 7 are reserved and zero. The canceled bit is valid only with archived lifecycle. A source Change status `abandoned` maps to archived lifecycle, canceled set, and the existing superseded bit clear. It does not create another Change record or payload. For a legacy Remote Change format that demonstrably has no fetch time, conversion writes `fetched_at_s = 0`; a supplied valid value is preserved, and an invalid supplied value fails closed.

`published_remote_change_ordinal_plus1 = 0` means absent. Values `1..64` encode Remote Change ordinals `0..63`, so `C-01` is stored as `1` and `C-64` as `64`; values `65..255` are invalid. `reserved2` and `reserved3` must be zero. `LocalChangeRecord.local_meta` bit 0 is clear if and only if the ordinal field is zero. When it is set, the ordinal field and `published_at_s` are non-zero, and the owning Local Task must also have its published bit set. The Remote ordinal may differ from the Local `change_ordinal`; publication stores the exact Task-scoped ordinal returned by the Remote authority.

An offline converter reading the legacy explicit textual `published_change_id` parses the exact `C-01..C-64` suffix and writes its ordinal-plus-one value directly; it never fabricates a global Remote Change index. A converter reading the former fixed `u32 published_remote_change_index` interpretation must explicitly select that source schema and resolve the referenced Remote record to its owning Task and Task-scoped ordinal before writing this field. Missing, mismatched, or ambiguous publication authority fails closed.

Inline Change Lifecycle Fields

`base_line_index_plus1` and `archived_at_s` are required fixed fields in each 68-byte Local or Remote Change record. They are not payload, a separate record, or a separate file. `base_line_index_plus1` is required and references the exact `LineRecord` from which the Change was created. A Change never stores a duplicate base-Line name or textual Line ID. If `fork_snapshot_index_plus1` is non-zero, it references the exact live Snapshot observed as that base Line's head when the Change was created. A Snapshot's `line_index_plus1` is its immutable authoring-Line identity, not Line-head membership; it may differ from `base_line_index_plus1` after a Snapshot authored on another Line becomes the base Line's head and must not be rewritten or compared for equality. The public `forked_from_line` value is derived from the numeric base-Line identity.

`archived_at_s` is non-zero if and only if the Change lifecycle is archived, except that an archived Change converted from a legacy source format that demonstrably did not record archive time retains zero as unknown historical archive time. `ChangeRecord.change_meta` bit 7 distinguishes a superseded Change inside that archived lifecycle, while `ChangeRecord.change_state` bit 0 distinguishes cancellation. Archive and reopen each rewrite and durably commit one complete 68-byte Change record; lifecycle bits and `archived_at_s` change together, with no side-file ordering or intra-record status-last protocol. Recovery rejects a non-archived Change carrying a non-zero archive time. Native v0 archive writes require a non-zero archive time. Because no migration bit is added, readers, validators, and recovery accept archived plus zero as unknown legacy time. The public `review` state is active lifecycle plus `review_pending`; `landed_at` is derived from the successful Land record's `updated_at_s` and is not stored again on the Change.

Change creation resolves the normalized base-Line name by exact payload comparison, observes that Line's current head as the optional fork Snapshot, then appends the complete Change commit record. Recovery rejects a zero or missing base Line, a missing/ambiguous Line, a missing or tombstoned fork Snapshot, a lifecycle/time mismatch, or an invalid superseded bit for a committed Change. Recovery does not compare the fork Snapshot's authoring Line with the Change's base Line.

The immediately preceding active layout-1 representation with a 44-byte `change.bin` and ordinal-aligned 8-byte `change_lifecycle.bin` is offline conversion input only. A converter must receive that source representation explicitly, require both files to have the same record count, preserve each Change's complete 44-byte prefix, append the exact paired base-Line index and archive time, validate the resulting complete 52-byte file, and omit `change_lifecycle.bin` from the activated target. It must not infer a missing base Line or align rows by any textual identity.

An earlier pre-side-file layout-1 generation is also offline conversion input only when its declared source schema independently supplies the exact `base_line`. Conversion resolves that value to an authority-local `line_index`, requires any supplied `forked_from_line` to equal that base Line, and preserves the exact referenced live fork Snapshot without comparing or rewriting its authoring Line. It preserves a supplied valid source archive time and writes `archived_at_s = 0` only when the admitted source format demonstrably has no archive-time field. An invalid supplied time or any unavailable or ambiguous required non-time value fails closed before materializing the complete 68-byte Change file.

Worktree Cursor

```
WORKTREE_CURSOR_RECORD_SIZE = 28

WorktreeCursorRecord - .ait-worktree/cursor.bin:
u8  cursor_meta
u8  reserved0
u16 reserved1
u32 task_index
u32 selected_change_index_plus1
u32 pending_pre_land_target_snapshot_index_plus1
u32 pending_landed_snapshot_index_plus1
u64 updated_at_s
```

The cursor is local-only, disposable scratch state. It is not workflow history and must be excluded from content snapshots.

Line And Stash Records

```
LINE_RECORD_SIZE = 40

LineRecord - line.bin:
u8  line_meta
u8  reserved0
u16 line_name_len
u64 line_name_offset
u32 head_snapshot_index_plus1
u64 created_at_s
u64 updated_at_s
u64 archived_at_s
```

```
STASH_RECORD_SIZE = 8

StashRecord - stash.bin:
u8  stash_meta
u8  reserved0
u16 reserved1
u32 stash_snapshot_index
```

Stash storage is local-only. `stash_snapshot_index` must reference a content snapshot whose kind is `stash`. Stash deletion tombstones the record and never shifts later indexes.

Local Tag Records

```
TAG_RECORD_SIZE = 24

TagRecord - tag.bin:
u8  tag_meta
u8  reserved0
u16 payload_len
u64 payload_offset
u32 snapshot_index
u64 created_at_s
```

`tag_index` is the stable internal Tag identity and is the zero-based ordinal of `TagRecord` in `tag.bin`. The public Tag identity remains its normalized, non-empty UTF-8 name. `snapshot_index` references a live content Snapshot.

Tag creation appends its typed payload before its fixed record. Force-replacing an existing name preserves `tag_index`, appends a new payload, then updates the fixed record; an interrupted replacement may leave only an unreferenced payload. Deletion sets the tombstone bit and never deletes the referenced Snapshot or shifts a Tag ordinal. Re-creation of the same name clears the tombstone only while preserving the same `tag_index`.

Land Records

```
TASK_LAND_INDEX_RECORD_SIZE = 8

TaskLandIndexRecord - task_land_index.bin:
u32 latest_land_index_plus1
u16 land_count
u16 reserved0

CHANGE_LAND_INDEX_RECORD_SIZE = 8

ChangeLandIndexRecord - change_land_index.bin:
u32 latest_land_index_plus1
u16 land_count
u8  next_land_ordinal
u8  reserved0
```

```
LOCAL_LAND_RECORD_SIZE = 44

LocalLandRecord - land.bin:
u8  land_meta
u8  land_ordinal
u8  change_ordinal
u8  failure_kind
u32 change_index
u32 previous_task_land_index_plus1
u32 previous_change_land_index_plus1
u32 pre_land_target_snapshot_index_plus1
u32 landed_snapshot_index_plus1
u64 submitted_at_s
u64 updated_at_s
u32 target_line_index_plus1
```

```
SERVER_LAND_RECORD_SIZE = 48

ServerLandRecord - land.bin:
u8  land_meta
u8  land_ordinal
u8  change_ordinal
u8  failure_kind
u32 change_index
u32 patchset_index
u32 previous_task_land_index_plus1
u32 previous_change_land_index_plus1
u32 pre_land_target_snapshot_index_plus1
u32 landed_snapshot_index_plus1
u64 submitted_at_s
u64 updated_at_s
u32 target_line_index_plus1
```

Local Land authority is the stored snapshot pair. Server Land authority also stores the exact accepted Patchset. Land status updates that keep one `L-##` use status-last two-phase writes.

`land_ordinal` is unique in `(change_index, land_ordinal)`. Values `0..63` render as `L-01..L-64` under the full owning Change identity. A Server Land's `patchset_index` is the immutable Patchset accepted by that attempt; it does not make Land identity Patchset-scoped. One Change may therefore retain `L-01` against one Patchset and `L-02` against a later Patchset. `ChangeLandIndexRecord` owns next-ordinal allocation and its latest/previous links follow Change-scoped Land ordinal. `TaskLandIndexRecord` and `previous_task_land_index_plus1` retain physical Task inventory only.

Land mode is stored in `land_meta` bits 5 through 6. Source `direct`, `merge`, and `ff-only` map to the fixed encodings below; an unknown mode fails closed. The source result object does not become payload. Target-Line identity and the pre-land/landed Snapshot values come from the fixed Land record; a blocked result's `blocker_class` maps to `failure_kind`. `BASE_STALE` and `POLICY_BLOCKED` map to `base_stale` and `policy_blocked` respectively. Any unrecognized blocker or result value that is not exactly derivable from that fixed record fails closed.

Native v0 Land submission requires non-zero `submitted_at_s`. For an offline legacy conversion only, a landed Change whose source authority records the exact landed result but never recorded submission time may materialize its first and only successful Land with `land_ordinal = 0`, both previous-Land links zero, `submitted_at_s = 0`, and `updated_at_s` equal to the preserved source `landed_at`. Zero means unknown historical submission time; a converter must not substitute Change creation/update time or conversion time. This rule does not permit reconstruction when target Line, landed Snapshot, accepted Patchset where required, or landed time is missing or ambiguous, and native v0 writers must never create a new Land with zero submission time.

Inline Land Target Line Authority

The owning Land record's `target_line_index_plus1` is required and references the exact target `LineRecord` selected when the Land is submitted. It is fixed schema, not payload. A Land never stores a duplicate target-Line name or textual Line ID.

When present, `pre_land_target_snapshot_index_plus1` references the exact live Snapshot observed as the target Line's head before its update, while `landed_snapshot_index_plus1` references the exact live Snapshot installed as that Line's new head. A referenced Snapshot's `line_index_plus1` remains its immutable authoring-Line identity and may differ from `target_line_index_plus1`; Land never relabels it or compares those Line indexes for equality. A landed Change's public `target_line` and `landed_at` values derive from its successful Land's target Line and `updated_at_s`;

queued, running, blocked, failed, canceled, and updating Land attempts retain their target Line even when neither Snapshot is yet present.

The admitted legacy server format may retain more than one succeeded Land for one Change and may append a blocked, failed, canceled, or updating attempt after an earlier success. Conversion preserves every Land row and its exact accepted Patchset. If any surviving Land succeeded, the Change lifecycle is normalized to landed regardless of the duplicated source Change status. The succeeded Land with the greatest `land_ordinal` supplies the public target Line and landed time; later non-succeeded attempts remain immutable history and do not undo the completed land. A selected/current Patchset created after that success remains independent mutable Change authority and need not equal the accepted Patchset. A source Change that claims landed but has no provable successful Land follows only the explicit reconstruction or named-omission rules below; it never receives a fabricated Land implicitly.

Legacy Change `landed_at` is a non-authoritative duplicated projection when exact Land authority exists. Conversion validates the supplied value as a timestamp but does not require it to equal any Land time and does not store it; the authoritative public time is the latest succeeded Land's `updated_at_s`. Likewise, the Change's mutable selected-Patchset pointer and a historical Land's exact accepted Patchset are independent authorities after submission. Conversion preserves both and does not require them to remain equal.

Land submission resolves the normalized Line name by exact payload comparison and appends and fsyncs one complete Land commit record containing the required target-Line reference. Recovery rejects a zero/missing target Line, a missing or tombstoned referenced Snapshot, or a status/Snapshot-presence disagreement for a committed Land. Recovery does not compare a referenced Snapshot's authoring Line with the Land target Line.

The immediately preceding layout-1 representation with 32-byte Local or 36-byte Server Land records plus ordinal-aligned 4-byte `land_target_line.bin` is offline conversion input only. Conversion requires matching headers and exact record counts, appends each exact target-Line value to its unchanged Land prefix, validates every widened record, omits the retired side file, and fails closed on zero, missing, extra, misaligned, or ambiguous input. The source files are not modified.

An earlier pre-side-file layout-1 generation without `land_target_line.bin` is also offline conversion input only. Conversion resolves the exact source `target_line` to an authority-local `line_index`, preserves each exact stored live Snapshot reference without comparing or rewriting its authoring Line, and fails closed if the target or a referenced Snapshot is unavailable or ambiguous before materializing the complete widened Land record.

BINARY DB V0 SCHEMA

Binary DB v0: Snapshot and Content Records

Expand Snapshot ancestry, Snapshot links, Tree, Blob, object-pack, manifest, and optional file-cache layouts.

AUDIENCE Storage implementers and recovery tooling authors

https://ait-native.dev/technical/v1.1.1/binary-db/content/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

This chapter defines immutable content identity and its physical storage relationships. Ordered parent edges and normalized Tree ranges are authority, while explicitly named cache families remain rebuildable or optional as stated below.

Content Snapshot Records

```
CONTENT_SNAPSHOT_RECORD_SIZE = 88

ContentSnapshotRecord - snapshot.bin:
u8  snapshot_meta
u8  history_flags
u16 payload_len
u64 payload_offset
u64 snapshot_hash48
u32 parent_snapshot_index_plus1
u32 root_tree_pack_index_plus1
u32 root_entry_ordinal
u32 line_index_plus1
u8  manifest_hash[32]
u32 file_count
u64 total_bytes
u64 created_at_s
```

The canonical public Snapshot ID is `SNP-` followed by twelve uppercase hex characters. `snapshot_hash48` stores that suffix in its low 48 bits; its high 16 bits must be zero. Non-standard Snapshot IDs are not representable in this layout and must be rejected.

`history_flags` is fixed Snapshot schema, not payload. Bit 0 marks an imported remote-head history boundary: the source Snapshot had at least one parent, but the local import intentionally did not materialize ancestry before that head. Such a Snapshot is not a true root. Bits 1 through 7 are reserved and must be zero.

Ordered Snapshot Parent Records

```
SNAPSHOT_PARENT_EDGE_RECORD_SIZE = 12

SnapshotParentEdgeRecord - snapshot_parent_edge.bin:
u32 child_snapshot_index
u32 parent_snapshot_index
u16 parent_ordinal
u16 flags
```

When `ContentSnapshotRecord.snapshot_meta` bit 4 is set, the ordered rows in `snapshot_parent_edge.bin` are the sole parent authority for that Snapshot. The fixed `parent_snapshot_index_plus1` field is then only a verified cache of ordinal zero: it is zero for a true root or an imported remote-head history boundary and otherwise must equal the ordinal-zero edge's `parent_snapshot_index + 1`. Snapshot payload bytes never carry parent indexes.

An admitted parent set has at most 1,024 rows. A non-empty set starts at ordinal zero and uses contiguous ordinals without duplicates. Ordinal zero is the primary parent used by explicit first-parent traversal. A child may not reference itself, a missing Snapshot, a tombstoned Snapshot, or form a cycle. Every `flags` value is zero in v0. Parent order is immutable and participates in the Snapshot manifest identity.

A Snapshot with `history_flags` bit 0 set must also have `parent_edges_authority` set, `parent_snapshot_index_plus1 = 0`, and no parent edge rows. Parent and first-parent traversal stops at that Snapshot. Export, reachability, and ancestry proofs must preserve the boundary distinction and must not report the locally

truncated graph as complete history. Setting the flag on a source root, on a Snapshot with any local parent, or without evidence that the source had a parent is corruption and fails closed.

A record with bit 4 clear uses the original fixed pointer as its complete zero-or-one-parent authority and has no parent-edge rows. This state exists only to admit a predecessor layout-1 generation as offline conversion input. New writes and newly activated generations use edge authority for every live Snapshot, including roots. Mixed authority for one Snapshot, a payload parent extension, non-contiguous ordinals, pointer/edge disagreement, or an edge for a bit-4-clear child is corruption and fails closed.

Snapshot authority is a DAG forest, not a single-root tree. An authority with zero live Snapshots is valid. Every live non-boundary Snapshot must reach one proven true root, but distinct components may reach distinct true roots. A predecessor bit-4-clear record whose complete fixed pointer is zero proves a true root; conversion writes edge authority with no parent rows and keeps `history_flags` clear. Neither record position, Line membership, component size, nor current-head status permits merging components or converting a true root into a remote-head history boundary.

Task Snapshot Link Records

```
TASK_SNAPSHOT_INDEX_RECORD_SIZE = 8

TaskSnapshotIndexRecord - task_snapshot_index.bin:
u32 latest_snapshot_link_index_plus1
u16 snapshot_count
u8  next_snapshot_ordinal
u8  reserved0

CHANGE_SNAPSHOT_INDEX_RECORD_SIZE = 8

ChangeSnapshotIndexRecord - change_snapshot_index.bin:
u32 latest_snapshot_link_index_plus1
u16 snapshot_count
u16 reserved0
```

```
AUTHORITATIVE_SNAPSHOT_LINK_RECORD_SIZE = 40

LocalTaskSnapshotLinkRecord / ServerTaskSnapshotLinkRecord - snapshot_link.bin:
u8  link_meta
u8  snapshot_ordinal
u16 payload_len
u64 payload_offset
u32 task_index
u32 change_index_plus1
u32 content_snapshot_index
u32 previous_task_snapshot_link_index_plus1
u32 previous_change_snapshot_link_index_plus1
u64 created_at_s
```

```
REMOTE_MIRROR_SNAPSHOT_LINK_RECORD_SIZE = 52

RemoteMirrorTaskSnapshotLinkRecord - snapshot_link.bin:
u8  link_meta
u8  remote_meta
u16 reserved1
u8  snapshot_ordinal
u8  reserved0
u16 payload_len
u64 payload_offset
u32 task_index
u32 change_index_plus1
u32 content_snapshot_index
u32 previous_task_snapshot_link_index_plus1
u32 previous_change_snapshot_link_index_plus1
u64 created_at_s
u64 fetched_at_s
```

`snapshot_ordinal` uses the full `u8` range. Values `0..255` render as `S-01..S-256`. A writer must reject the next link after ordinal 255 and must not wrap, clamp, reuse, or invent an extended Snapshot handle.

Tree Records

```
TREE_PACK_RECORD_SIZE = 32

TreePackRecord - tree_pack.bin:
u8  pack_meta
u8  pack_format_kind
u16 pack_hash_hi16
u32 pack_hash_lo32
u32 first_tree_index
u32 tree_count
u64 total_bytes
u64 created_at_s
```

```
TREE_RECORD_SIZE = 20

TreeRecord - tree.bin:
u8  tree_meta
u8  reserved0
u32 pack_entry_ordinal
u32 entry_count
u8  tree_hash80[10]
```

```
TREE_ENTRY_RANGE_RECORD_SIZE = 4

TreeEntryRangeRecord - tree_entry_range.bin:
u32 first_entry_index
```

```
TREE_ENTRY_RECORD_SIZE = 16

TreeEntryRecord - tree_entry.bin:
u8  entry_meta
u8  name_len
u16 mode_bits
u64 name_offset
u32 target_index
```

Tree entry names are individual normalized UTF-8 path segments, not full paths, and are limited to 255 bytes. Blob entries target `blob.bin`; Tree entries target `tree.bin`.

`TreeRecord.pack_entry_ordinal` is the exact physical entry ordinal inside the Tree pack archive that supplies this Tree's raw payload. For a Tree Pack with `sparse_physical_ordinals` clear, every Tree in its logical range satisfies `pack_entry_ordinal = tree_index - first_tree_index` and the ordinal is less than `tree_count`. When the bit is set, the ordinal is instead the exact sparse physical archive locator and need not equal the logical offset or be less than the logical `tree_count`; archive bounds, decoded Tree ID, and decoded entry count are still verified.

`TreeEntryRangeRecord` is fixed schema, not payload. Its record ordinal is the `tree_index`, and at every stable activation boundary `tree_entry_range.bin` has exactly as many records as `tree.bin`. Together with the owning `TreeRecord.entry_count`, `first_entry_index` names that Tree's authoritative contiguous normalized rows in `tree_entry.bin`. For an empty Tree, the canonical `first_entry_index` is the current `tree_entry.bin` record count. The range must remain within `tree_entry.bin`; new writes append normalized entry rows in Tree order and never overlap a committed Tree's range.

The normalized range is authoritative for metadata traversal. The physical pack locator is authoritative for locating and verifying the archived raw Tree payload. Names, modes, target kinds, targets, Tree ID, and `entry_count` must agree across the normalized range and decoded pack entry; disagreement is corruption and fails closed.

Tree creation appends and fsyncs its normalized entry rows and range dependency before appending the Tree commit record. Recovery ignores or truncates orphan entry/range dependencies, rejects a missing range for a committed Tree, and validates every committed range and physical pack locator.

A predecessor layout-1 generation without `tree_entry_range.bin` is offline conversion input only. Conversion preserves the existing `pack_entry_ordinal`, reconstructs and verifies every normalized range from the pack payload, and materializes the complete side file before activation.

Blob And Object-Pack Records

```
OBJECT_PACK_RECORD_SIZE = 32

ObjectPackRecord - object_pack.bin:
u8  pack_meta
u8  pack_format_kind
u16 pack_hash_hi16
u32 pack_hash_lo32
u32 first_member_index
u32 member_count
u64 total_bytes
u64 created_at_s
```

```
OBJECT_PACK_MEMBER_RECORD_SIZE = 16

ObjectPackMemberRecord - object_pack_member.bin:
u8  member_meta
u8  delta_chain_depth
u16 reserved0
u32 pack_index
u32 blob_index
u32 base_blob_index_plus1
```

```
BLOB_RECORD_SIZE = 64

BlobRecord - blob.bin:
u8  blob_meta
u8  hash_kind
u16 reserved0
u64 size_bytes
u32 pack_member_index_plus1
u64 created_at_s
u64 pruned_at_s
u8  sha256[32]
```

Blob SHA-256 is stored as 32 raw bytes. Public `BLB-*` identity renders from the first 80 bits. Object and Tree pack records store their 48-bit public-ID suffix as `(pack_hash_hi16, pack_hash_lo32)`.

For admitted legacy conversion, multiple source Blob rows with the same full 32-byte SHA-256 collapse to one target Blob only when `blob_meta`, normalized `hash_kind`, `reserved0`, `size_bytes`, and `pruned_at_s` agree. Distinct pack-member pointers are physical copies of the same content and every member, base-Blob, Tree-entry, and rebuilt-index reference is densely remapped. The target `created_at_s` is the minimum valid non-zero source creation time. A zero time, metadata disagreement, size disagreement, prune disagreement, hash prefix collision with different complete SHA-256 bytes, or byte/hash mismatch fails closed. This normalization adds no Blob alias row or mapping file.

Optional Manifest And File Cache

These records are disposable local acceleration and are never authoritative:

```
SNAPSHOT_FILE_INDEX_RECORD_SIZE = 4

SnapshotFileIndexRecord - snapshot_file_index.bin:
u32 first_file_index_plus1
```

```
SNAPSHOT_FILE_RECORD_SIZE = 24

SnapshotFileRecord - snapshot_file.bin:
u64 path_hash64
u64 path_offset
u32 tree_entry_index
u16 path_len
u16 reserved0
```

The authoritative path remains `snapshot.bin -> tree_pack.bin/tree.bin/ tree_entry.bin -> blob.bin`. Readers must remain correct when every optional manifest/file cache is absent.

BINARY DB V0 SCHEMA

Binary DB v0: Git Interoperability Records

Expand the optional Git repository, generation, identity, commit, file, ref, Tag, and checkpoint mapping layouts.

AUDIENCE Git interoperability implementers

https://ait-native.dev/technical/v1.1.1/binary-db/git/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

Git records form an optional import/export mapping authority around AIT content; they do not replace primary workflow or content authority. The layouts retain exact source bytes where specified and fail closed on unsupported object or identity representations.

Local Git Import/Export Records

Git interoperability is an optional local escape-hatch authority. Its fixed records preserve the landed `git-identity-map/v1` and `git-interop-checkpoint/v1` semantics without making Git the primary AIT authority. Every Git Object ID field in v0 is exactly 20 raw SHA-1 bytes. Another Git object format fails before any AIT or Git ref mutation; a writer must not widen these fields under `layout_id = 1`.

```
GIT_REPOSITORY_RECORD_SIZE = 28

GitRepositoryRecord - git_repository.bin:
u8  repository_meta
u8  object_format_kind
u16 reserved0
u32 identity_len
u64 identity_offset
u8  fingerprint96[12]
```

`fingerprint96` stores the 24 hexadecimal digits of the current `GSR-*`, `GTR-*`, or `ASR-*` repository fingerprint as 12 raw bytes. The prefix is reconstructed from `repository_meta`. The typed identity payload stores the same canonical source identity, target path, or AIT repository identity used to calculate that fingerprint.

```
GIT_GENERATION_RECORD_SIZE = 20

GitGenerationRecord - git_generation.bin:
u8  generation_meta
u8  reserved0
u16 reserved1
u32 source_repository_index
u32 target_repository_index_plus1
u8  generation_hash64[8]
```

`generation_hash64` stores the 16 hexadecimal digits of `GIT-IMP-*` or `GIT-EXP-*` as eight raw bytes. Import has one Git source repository and no stored target repository. Export has an AIT source repository and one Git target repository. A generation is immutable; it groups mappings created from one import source state or one deterministic export plan. This content generation is not a Binary DB transaction or activation generation.

```
GIT_IDENTITY_RECORD_SIZE = 24

GitIdentityRecord - git_identity.bin:
u8  identity_meta
u8  reserved0
i16 timezone_offset_minutes
u32 payload_len
u64 payload_offset
i64 timestamp_s
```

An identity record preserves the current `raw`, `name`, `email`, `timestamp`, and `timezone` fields. `timestamp_s` is the signed Git identity timestamp; `timezone_offset_minutes` is its parsed numeric offset. The raw identity bytes remain available for lossless imported-object evidence.

```
GIT_COMMIT_MAPPING_RECORD_SIZE = 96

GitCommitMappingRecord - git_commit_mapping.bin:
u8  mapping_meta
u8  reserved0
u16 parent_count
u32 generation_index
u32 snapshot_index
u32 author_identity_index
u32 committer_identity_index
u32 first_parent_mapping_index_plus1
u32 first_file_mapping_index_plus1
u32 file_count
u32 payload_len
u64 payload_offset
u8  git_object_id[20]
u8  git_tree_object_id[20]
u32 lfs_pointer_count
u64 created_at_s
```

The referenced Snapshot supplies the AIT root Tree locator and textual Snapshot identity; neither is duplicated here. `parent_count` is bounded by the Snapshot limit of 1,024. Parent and file mapping ranges are contiguous; their `first_*_index_plus1` field is zero exactly when the corresponding count is zero. Commit mapping records are immutable and append-only.

```
GIT_COMMIT_PARENT_RECORD_SIZE = 28

GitCommitParentRecord - git_commit_parent.bin:
u32 commit_mapping_index
u32 snapshot_parent_edge_index
u8  parent_git_object_id[20]
```

The referenced Snapshot parent edge owns `parent_ordinal` and the AIT parent Snapshot index. It must name the same child as the owning commit mapping's `snapshot_index`. Parent mapping range order must equal edge ordinal order, so Git commit parent order and Snapshot DAG order cannot diverge.

```
GIT_FILE_MAPPING_RECORD_SIZE = 52

GitFileMappingRecord - git_file_mapping.bin:
u8  git_object_type_kind
u8  reserved0
u16 reserved1
u32 commit_mapping_index
u32 entry_ordinal
u32 tree_entry_index
u32 git_mode
u32 path_len
u64 path_offset
u8  git_object_id[20]
```

The Tree entry owns the AIT mode, target Tree/Blob index, and content identity; those values are not duplicated. `git_mode`, the Git object type and Object ID, and the exact Git path bytes preserve the Git-side tree mapping. Entry ordinals are contiguous inside one commit mapping range.

```
GIT_REF_MAPPING_RECORD_SIZE = 80

GitRefMappingRecord - git_ref_mapping.bin:
u8  ref_meta
u8  git_object_type_kind
u16 reserved0
u32 generation_index
u32 snapshot_index_plus1
u32 target_index_plus1
u32 symbolic_target_ref_mapping_index_plus1
u32 ref_name_len
u64 ref_name_offset
u8  git_object_id[20]
u8  expected_previous_git_object_id[20]
u64 created_at_s
```

Branch mappings use `target_index_plus1` as a numeric `line_index + 1`; Tag mappings use it as `tag_index + 1`. They never store AIT Line ID, Line name, Tag name, `ait_kind`, `ait_name`, or `ait_identity`. A symbolic `HEAD` record references the mapped target ref record numerically instead of duplicating a Line name. `expected_previous_git_object_id` is meaningful only when its meta bit is set and records export compare-and-swap evidence; otherwise all 20 bytes are zero.

```
GIT_TAG_MAPPING_RECORD_SIZE = 48

GitTagMappingRecord - git_tag_mapping.bin:
u8  tag_mapping_meta
u8  reserved0
u16 reserved1
u32 git_ref_mapping_index
u32 payload_len
u64 payload_offset
u8  peeled_commit_git_object_id[20]
u64 created_at_s
```

The referenced Git ref mapping supplies the Git ref name, Git object ID/type, Snapshot index, and numeric AIT Tag index. A lightweight Tag has Git object type `commit` and empty raw Tag bytes. An annotated Tag has object type `tag` and preserves its message and complete raw Tag object in the typed payload.

```
GIT_OPERATION_CHECKPOINT_RECORD_SIZE = 44

GitOperationCheckpointRecord - git_operation_checkpoint.bin:
u8  checkpoint_meta
u8  reserved0
u16 reserved1
u32 generation_index
u8  operation_hash64[8]
u8  plan_hash96[12]
u32 next_commit_index
u32 next_ref_index
u64 updated_at_s
```

`operation_hash64` stores the 16 hexadecimal digits of `GIO-IMPORT-*` or `GIO-EXPORT-*`; `plan_hash96` stores the 24 hexadecimal digits of `GIP-*` or `GEP-*`. The two next indexes are resumable plan cursors. The checkpoint is updated in place with `checkpoint_meta` written last as the state commit marker. A completed operation result is deterministically rebuilt from its generation and immutable mapping rows; no opaque result JSON is authoritative.

Mapping `contract`, textual `record_id`, textual AIT IDs, Base64 wrappers, and the duplicated JSON arrays `parent_*_ids` and `file_modes` are not stored. Public `GIM-*` record identity remains derived by the landed `git-identity-map/v1` logical canonicalization, excluding `created_at` and the record ID itself. Commit, ref, and Tag mapping physical ordinals are their exact append order; no AIT-generated subsecond ordering timestamp is persisted. The signed `GitIdentityRecord.timestamp_s` and timezone are retained separately because they are exact Git identity data. `identity_source`, `published_remote_name`, and a new planning-state field are not part of Git interop. Git import fabricates no Task or policy lineage; the existing `TaskRecord.task_meta` planned bit remains the complete binary planning state.

BINARY DB V0 SCHEMA

Binary DB v0: Server and Policy Records

Expand Patchset, attestation, Actor, review, policy, waiver, Plan, Repository registry, and Worker Job authority.

AUDIENCE Server, runner, policy, and recovery implementers

https://ait-native.dev/technical/v1.1.1/binary-db/server/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

This chapter covers repository-scoped remote workflow authority and the separate installation-scoped Repository registry. Numeric indexes are meaningful only in their declared roots; Worker Job identity and references stay inside the routed Repository authority.

Server Patchset Records

```
TASK_PATCHSET_INDEX_RECORD_SIZE = 8

TaskPatchsetIndexRecord - task_patchset_index.bin:
u32 latest_patchset_index_plus1
u16 patchset_count
u16 reserved0

CHANGE_PATCHSET_INDEX_RECORD_SIZE = 8

ChangePatchsetIndexRecord - change_patchset_index.bin:
u32 latest_patchset_index_plus1
u16 patchset_count
u8  next_patch_ordinal
u8  reserved0
```

```
SERVER_PATCHSET_RECORD_SIZE = 65

ServerPatchsetRecord - patchset.bin:
u8  patchset_meta
u8  patch_ordinal
u8  change_ordinal
u8  reserved0
u32 change_index
u32 previous_task_patchset_index_plus1
u32 previous_change_patchset_index_plus1
u32 base_snapshot_index
u32 revision_snapshot_index
u64 created_at_s
u64 ci_completed_at_s
u32 ci_run_seq
u16 ci_selected_suite_count
u16 ci_suite_result_count
u16 ci_blocking_failure_count
u8  ci_status_bits
u64 summary_offset
u16 summary_len
u32 ci_worker_job_index_plus1
```

The first 51 bytes are the complete fixed Patchset/compact-CI region: its first 32 bytes are the widened Patchset identity prefix and its corrected compact CI tail is exactly 19 bytes. Those regions are the zero-extended forms of the `u32-time-v0` predecessor's 28-byte identity prefix and 15-byte compact-CI tail. The appended summary locator is exactly ten bytes. The final Worker Job locator is exactly four bytes, producing the complete 65-byte record. None of these regions has implicit padding. `summary_len` is in `1..65535`, and `summary_offset` points to exactly that many non-empty UTF-8 bytes in `patchset_summary_payload.bin`. Patchset summary is the human-authored description of this reviewable Patchset; it is not Task intent, Snapshot message, Tree diff statistics, Review text, or Policy output.

`patch_ordinal` is unique in `(change_index, patch_ordinal)`. Values `0..63` render as `P-01..P-64` under the full owning Change identity. `ChangePatchsetIndexRecord.next_patch_ordinal` is the one-past-greatest allocated ordinal and rejects allocation after 64; deletion or omission never reuses an ordinal.

`ChangePatchsetIndexRecord.latest_patchset_index_plus1` and `previous_change_patchset_index_plus1` follow Change-scoped Patchset ordinal. `TaskPatchsetIndexRecord` is physical inventory only: its latest pointer and each `previous_task_patchset_index_plus1` follow committed physical Patchset record order and do not allocate or imply a Task-scoped Patchset ordinal. Recovery rebuilds the Task inventory from committed record order and rebuilds each Change chain, count, and next ordinal from Change-scoped identity.

Patchset creation appends and fsyncs the summary bytes before appending the fixed Patchset commit record. Patchset identity, owner, ordinals, Snapshot references, creation time, and summary locator/bytes are immutable. Compact CI fields and `ci_worker_job_index_plus1` may change only by complete-record replacement under the declared Worker Job mutation boundary. An interrupted append may leave only unreferenced trailing summary bytes, which recovery ignores. A committed missing, empty, overlapping, or invalid UTF-8 summary range is corruption.

`ci_completed_at_s` is a non-negative `u64` Unix-second timestamp. `ci_run_seq = 0` means no CI run has started. A non-zero `ci_run_seq` with `ci_completed_at_s = 0` represents an in-flight run and requires all status and count fields to be zero. Completed evidence requires a non-zero completion time, a non-zero run sequence, and a non-`none` overall status.

`ci_worker_job_index_plus1 = 0` means the Patchset has no selected Worker-Job-backed CI run; this includes legacy compact CI evidence and an inline CI run. A non-zero value resolves to `index + 1` in `worker_job.bin` under the same server Repository authority as this Patchset. It must target a non-tombstoned Job whose exact type is `patchset.ci` and whose fixed `patchset_index_plus1` resolves back to this exact Patchset. No Repository ID or Repository index participates in that same-root relationship.

The locator selects the greatest committed `worker_job_index` for this Patchset. A later rerun appends a new Job and replaces the complete Patchset record with that new local index; older Jobs remain Job history but cannot overwrite compact CI evidence after they are superseded. At terminal completion, compact CI fields may be replaced from a Job result only after revalidating that the completing Job is still the selected locator target.

```
ServerPatchsetRecord.ci_status_bits:
bits 0..1 overall_status
bits 2..3 tests_status
bits 4..5 lint_status
bits 6..7 reserved = 0

Each two-bit status:
00 none
01 pass
10 fail
11 error
```

`ci_suite_result_count` must not exceed `ci_selected_suite_count`, and `ci_blocking_failure_count` must not exceed `ci_suite_result_count`. Component statuses require completed overall evidence. These fields store only compact Patchset CI evidence. Queue state, attempts, fixed outcomes, and retry-error categories remain in the same Repository's fixed Worker Job family. Lease credentials, request materialization, detailed results, logs, artifacts, and scheduler configuration remain outside Binary DB v0.

Source `author_mode`, the published/superseded portion of `publish_state`, and the pending/non-pending Policy cache state are encoded in `patchset_meta` as defined below. A source selected-for-landing projection is normalized under the Remote Change pointer rules and is not Patchset state. Source `diff_stats` is not stored: it is recomputed by exact comparison of the base and revision Snapshot Trees. Except for the exact named legacy all-zero gate under Bin-to-Bin Conversion Acceptance, a converter must reject a supplied value that disagrees with that comparison. A non-pending `evaluation_state` is the latest live Policy Decision kind; the Patchset pending bit distinguishes a newly created or invalidated cache from an older non-pending decision. No Patchset JSON payload is authoritative.

Patchset storage is server-authoritative. Local workflow authority defines no Patchset files.

Server Attestation Records

```
TASK_ATTEST_INDEX_RECORD_SIZE = 8

TaskAttestIndexRecord - task_attest_index.bin:
u32 latest_attest_index_plus1
u16 attest_count
u8  next_attest_ordinal
u8  reserved0

PATCHSET_ATTEST_INDEX_RECORD_SIZE = 8

PatchsetAttestIndexRecord - patchset_attest_index.bin:
u32 latest_attest_index_plus1
u16 attest_count
u16 reserved0
```

```
SERVER_ATTEST_RECORD_SIZE = 24

ServerAttestationRecord - attest.bin:
u8  attest_meta
u8  attest_ordinal
u8  patch_ordinal
u8  change_ordinal
u32 patchset_index
u32 previous_task_attest_index_plus1
u32 previous_patchset_attest_index_plus1
u64 created_at_s
```

The four finite Attestation requirement values are stored in `attest_meta` bits 3 through 6, never payload. A source compact Attestation whose time and requirement bits are all zero is absent and emits no v0 record. A present source compact Attestation is Change-scoped mutable source state; the legacy format does not persist selected-Patchset history at `attested_at_s`. While holding one coherent locked source snapshot, conversion resolves the row through its owning Change's authoritative current `selected_patchset_number` and emits one v0 Attestation whose `patchset_index` references that exact Patchset. The ordinal-to-target-index lookup is converter-local state and creates no historical selection field, record, bin, or payload. This lookup does not make current Change selection transient: the same source pointer independently populates the existing `RemoteChangeRecord.selected_patchset_index_plus1` authority defined above. A missing, invalid, non-owned, or non-unique current selected pointer fails closed. Any source Attestation author mode must equal the bound Patchset author mode because v0 does not duplicate it.

Actor And Review Records

```
ACTOR_RECORD_SIZE = 36

ActorRecord - actor.bin:
u8  actor_meta
u8  reserved0
u16 payload_len
u64 payload_offset
u64 actor_key_hash
u64 created_at_s
u64 last_seen_at_s
```

For bin-to-bin conversion, this existing fixed record and the existing `ActorPayload` are the complete Actor schema; no conversion-only Actor bin or generic payload exists. Each distinct non-empty source reviewer identity is stored byte-for-byte as `ActorPayload.user_name_bytes`, with `user_id_len = 0`, `email_len = 0`, and empty memo. The converter does not trim, case-fold, or parse a display string such as `Name <email>` into invented components. The Actor kind is `unknown` unless a structured source field proves another kind. A non-empty identity supplied through `requested_groups` is structured team evidence and uses kind `team`. Equal kind plus equal identity bytes reuse one Actor; different bytes remain different Actors.

The required payload is identity authority and collision evidence, not an optional Review annotation. Its length must fit the existing `u8 user_name_len` and `u16 payload_len`; overflow or invalid UTF-8 fails closed. `actor_key_hash` is FNV-1a-64 over the exact `user_name_bytes`:

```
actor_key_hash = fnv1a64(user_name_bytes)
offset_basis   = 0xcbf29ce484222325
prime          = 0x00000100000001b3
```

The `ActorLookupIndexRecord` may return hash-collision candidates. A lookup accepts a candidate only after exact `actor_kind` and `ActorPayload` byte comparison; the hash alone never merges identities. `created_at_s` is the minimum source `ReviewRecord.created_at_s` that references the Actor, and `last_seen_at_s` is the maximum. When the admitted legacy source format has no Review time, both fields are zero, meaning unknown historical times. Supplied valid times are preserved for the derivation and invalid supplied times fail closed; conversion time is never substituted.

```
TASK_REVIEW_INDEX_RECORD_SIZE = 8

TaskReviewIndexRecord - task_review_index.bin:
u32 latest_review_index_plus1
u16 review_count
u16 reserved0

PATCHSET_REVIEW_INDEX_RECORD_SIZE = 8

PatchsetReviewIndexRecord - patchset_review_index.bin:
u32 latest_review_index_plus1
u16 review_count
u8  next_review_ordinal
u8  reserved0
```

```
SERVER_REVIEW_RECORD_SIZE = 40

ServerReviewRecord - review.bin:
u8  review_meta
u8  review_ordinal
u8  patch_ordinal
u8  change_ordinal
u32 actor_index_plus1
u32 patchset_index
u32 previous_task_review_index_plus1
u32 previous_patchset_review_index_plus1
u64 payload_offset
u16 payload_len
u16 reserved0
u64 created_at_s
```

Review action variants use one base action plus the fixed modifiers in `review_meta`. The exact source mappings are:

```
request              -> request
comment              -> comment
task_comment         -> comment + task_lane
code_review_summary  -> comment + code_review_summary
approve              -> approve
task_approve         -> approve + task_lane
request_changes      -> request_changes
task_request_changes -> request_changes + task_lane
defer                -> comment + defer
task_defer           -> comment + task_lane + defer
dismiss              -> dismiss
```

Any other spelling or invalid base/modifier combination fails closed. The independent `blocking` bit is preserved for every mapping. A normal Review's `actor_index_plus1` references its reviewer Actor. A review request with exactly one `requested_groups` value uses action `request` and references the corresponding team Actor; the active v0 schema cannot encode multiple groups in one Review. Its source `reviewer` may be absent or may contain the exact same UTF-8 bytes as that sole non-empty group. In the equal case it is only a redundant legacy projection: conversion creates or reuses the one team Actor and stores one `actor_index_plus1`; it does not create a second Actor or Review. A converter presented with a different reviewer, an empty identity, zero groups, or more than one requested group for a request fails closed instead of inventing Review rows or payload. Review comment or request-note text remains in `ReviewPayload`.

`review_ordinal` is unique in `(patchset_index, review_ordinal)`. Values `0..63` render as `R-01..R-64` under the full owning Patchset identity. An admitted legacy Change-scoped Review ID retains its exact numeric suffix when it moves under that Review's exact Patchset; gaps inside one Patchset are valid, but duplicate ordinals are not. `PatchsetReviewIndexRecord` owns next-ordinal allocation and its latest/previous links follow Patchset-scoped Review

ordinal. `TaskReviewIndexRecord` and `previous_task_review_index_plus1` retain physical Task inventory only. Recovery rebuilds each Patchset chain, count, and next ordinal without renumbering source Review identities.

Policy And Waiver Records

```
TASK_POLICY_INDEX_RECORD_SIZE = 8

TaskPolicyIndexRecord - task_policy_index.bin:
u32 latest_policy_index_plus1
u16 policy_count
u16 reserved0

PATCHSET_POLICY_INDEX_RECORD_SIZE = 8

PatchsetPolicyIndexRecord - patchset_policy_index.bin:
u32 latest_policy_index_plus1
u16 policy_count
u8  next_policy_ordinal
u8  reserved0
```

```
POLICY_DECISION_RECORD_SIZE = 32

PolicyDecisionRecord - policy.bin:
u8  policy_meta
u8  policy_ordinal
u8  patch_ordinal
u8  change_ordinal
u32 patchset_index
u32 previous_task_policy_index_plus1
u32 previous_patchset_policy_index_plus1
u32 first_check_index_plus1
u16 check_count
u16 reserved0
u64 created_at_s
```

`policy_ordinal` is unique in `(patchset_index, policy_ordinal)`, not in the owning Task. Values `0..63` render as `K-01..K-64` under the full owning Patchset identity. `PatchsetPolicyIndexRecord.next_policy_ordinal` is the one-past-greatest allocated ordinal and rejects allocation after 64; deletion or tombstoning never reuses an ordinal. A Task may therefore own more than 64 Policy Decisions across multiple Patchsets, subject to its `u16 policy_count` capacity, while each Patchset remains capped at 64.

`PatchsetPolicyIndexRecord.latest_policy_index_plus1` and `previous_patchset_policy_index_plus1` follow Patchset-scoped Policy ordinal. `TaskPolicyIndexRecord` retains the complete Task inventory only: its latest pointer and each `previous_task_policy_index_plus1` follow committed physical Policy record order and do not allocate or imply a Task-scoped Policy ordinal. Recovery rebuilds the Task inventory from committed record order and rebuilds each Patchset chain, count, and next ordinal from Patchset-scoped identity.

```
POLICY_CHECK_RECORD_SIZE = 8

PolicyCheckRecord - policy_check.bin:
u8  check_kind
u8  check_status
u16 subject_ordinal
u32 detail_flags
```

The rows beginning at `first_check_index_plus1` preserve source check order and are the complete fixed Policy-check authority. `subject_ordinal` and `detail_flags` have these exact meanings:

```
check_kind 0..7:
  subject_ordinal = 0
  detail_flags = 0

check_kind 8 ci_rollout_phase:
  subject_ordinal = exact rollout phase in 0..65535
  detail_flags = 0

check_kind 9 ci_patchset_suite:
  subject_ordinal = one-based position of the exact suite ID in the
                    normalized UTF-8-byte-sorted exact Policy tuple catalog
  detail_flags bit 0 = blocking suite
  detail_flags bit 1 = informational suite
  detail_flags bits 2..31 = 0
```

`ci_rollout_phase` is the numeric rollout phase of CI enforcement policy. It is not Task, Change, Patchset, or Land lifecycle state, and it does not itself name a CI suite. For every admitted legacy phase-0 projection, the source check name is exactly `ci_rollout_phase`; conversion writes `check_kind = 8`, preserves the source check status, writes `subject_ordinal = 0`, and writes `detail_flags = 0`.

Legacy conversion accepts only the following exact `(catalog, blocking set, informational set, phase message)` tuples. Catalog comparison uses normalized UTF-8 byte order. A suite in the blocking set receives detail bit 0; a suite in the informational set receives detail bit 1.

```
catalog = [rust_core]
blocking = [rust_core]
informational = []
message = CI rollout phase 0 blocks `rust_core` and keeps none visible as non-blocking surfaces.

catalog = [full_repo_contract]
blocking = [full_repo_contract]
informational = []
message = CI rollout phase 0 blocks `full_repo_contract` and keeps none visible as non-blocking
  surfaces.

catalog = [full_repo_contract]
blocking = [full_repo_contract]
informational = []
message = CI rollout phase 0 blocks `full_repo_contract` and keeps none visible as non-blocking
  surfaces. Future promotions are modeled as phase1: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke]
blocking = [package_smoke, preflight, stable_smoke]
informational = [recent_regression]
message = CI rollout phase 0 blocks `package_smoke`, `preflight`, `stable_smoke` and keeps
  `recent_regression` visible as non-blocking surfaces.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, task_batch]
blocking = [package_smoke, preflight, stable_smoke]
informational = [recent_regression, task_batch]
message = CI rollout phase 0 blocks `preflight`, `stable_smoke`, `package_smoke` and keeps
  `recent_regression`, `task_batch` visible as non-blocking surfaces. Future promotions are
  modeled as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, task_batch, tg1_required]
blocking = [package_smoke, preflight, stable_smoke, tg1_required]
informational = [recent_regression, task_batch]
message = CI rollout phase 0 blocks `preflight`, `stable_smoke`, `package_smoke`, `tg1_required`
  and keeps `recent_regression`, `task_batch` visible as non-blocking surfaces. Future
  promotions are modeled as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, task_batch, tg1_required]
blocking = [package_smoke, preflight, stable_smoke]
informational = [recent_regression, task_batch, tg1_required]
message = CI rollout phase 0 blocks `preflight`, `stable_smoke`, `package_smoke` and keeps
  `recent_regression`, `task_batch`, `tg1_required` visible as non-blocking surfaces. Future
  promotions are modeled as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, task_batch, tg1_required]
blocking = [package_smoke, preflight, stable_smoke]
informational = [recent_regression, task_batch, tg1_required]
message = CI rollout phase 0 blocks `preflight`, `stable_smoke`, `package_smoke` and keeps
  `tg1_required`, `recent_regression`, `task_batch` visible as non-blocking surfaces. Future
  promotions are modeled as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, tg1_required]
blocking = [package_smoke, preflight, stable_smoke]
informational = [recent_regression, tg1_required]
message = CI rollout phase 0 blocks `package_smoke`, `preflight`, `stable_smoke` and keeps
  `recent_regression`, `tg1_required` visible as non-blocking surfaces. Future promotions are
  modeled as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.

catalog = [package_smoke, preflight, recent_regression, stable_smoke, tg1_required]
blocking = [package_smoke, preflight, stable_smoke, tg1_required]
informational = [recent_regression]
message = CI rollout phase 0 blocks `package_smoke`, `preflight`, `stable_smoke`, `tg1_required`
  and keeps `recent_regression` visible as non-blocking surfaces. Future promotions are modeled
  as phase1: `recent_regression`, `task_batch`, phase2: `full_repo`.
```

The phase check, exact suite-name checks, catalog, blocking/informational partition, and message must agree with one tuple. Labels remain derived presentation and are not persisted. A different or ambiguous representation fails closed; the converter never parses a phase number or suite assignment out of free-form text. Native v0 writers may use another numeric phase only with an explicit Policy configuration that supplies the fixed fields directly.

Exactly one of the two suite detail bits is set for `check_kind = 9`. Within one Policy, suite ordinals are non-zero, unique, complete, and no greater than that Policy's exact tuple catalog length. They are not bounded by the owning Patchset's `ci_selected_suite_count`: the Patchset field is compact evidence for one CI run, while immutable Policy rows retain historical evaluations that may precede that evidence or use a different admitted blocking/informational projection.

Conversion resolves a source `ci_patchset_suite_<suite-id>` name against the exact Policy tuple catalog; missing, duplicate, incomplete, or ambiguous suite identity fails closed. Policy check `label` and `message` are presentation projections and are not stored. An unknown check name/status fails closed rather than becoming text payload. Source `input_fingerprint` is cache metadata, not Policy authority; v0 recomputes it from the fixed Patchset, Attestation, Review, Waiver, CI, and Policy inputs and does not persist the source cache key.

```
TASK_WAIVER_INDEX_RECORD_SIZE = 8

TaskWaiverIndexRecord - task_waiver_index.bin:
u32 latest_waiver_index_plus1
u16 waiver_count
u8  next_waiver_ordinal
u8  reserved0

PATCHSET_WAIVER_INDEX_RECORD_SIZE = 8

PatchsetWaiverIndexRecord - patchset_waiver_index.bin:
u32 latest_waiver_index_plus1
u16 waiver_count
u16 reserved0
```

```
WAIVER_RECORD_SIZE = 44

WaiverRecord - waiver.bin:
u8  waiver_meta
u8  waiver_ordinal
u8  patch_ordinal
u8  change_ordinal
u32 patchset_index
u32 previous_task_waiver_index_plus1
u32 previous_patchset_waiver_index_plus1
u64 payload_offset
u16 payload_len
u16 rule_code
u64 created_at_s
u64 expires_at_s
```

Plan Records

```
PLAN_RECORD_SIZE = 48

PlanRecord - plan.bin:
u8  plan_meta
u8  reserved0
u16 payload_len
u64 payload_offset
u32 latest_revision_index_plus1
u32 published_plan_index_plus1
u32 published_latest_revision_index_plus1
u64 created_at_s
u64 updated_at_s
u64 published_at_s
```

```
PLAN_REVISION_RECORD_SIZE = 56

PlanRevisionRecord - plan_revision.bin:
u8  revision_meta
u8  reserved0
u16 payload_len
u16 revision_number
u16 item_count
u64 payload_offset
u32 plan_index
u32 previous_revision_index_plus1
u32 item_start_index
u32 published_revision_index_plus1
u32 root_tree_pack_index_plus1
u32 root_entry_ordinal
u64 created_at_s
u64 published_at_s
```

```
PLAN_ITEM_RECORD_SIZE = 16

PlanItemRecord - plan_item.bin:
u8  item_meta
u8  reserved0
u16 payload_len
u64 payload_offset
u32 line_number
```

Plan revision history is a linear chain:

```
PlanRecord.latest_revision_index_plus1
  -> PlanRevisionRecord.previous_revision_index_plus1
  -> ...
  -> 0
```

Task-to-Plan binding uses the Task record's numeric revision and item indexes. Exact taskability still requires comparison with `PlanItemPayload.plan_item_ref_bytes`.

Server-Global Repository Registry Authority

The server-global Repository registry is one installation-scoped Binary DB root. It is never nested inside a repository authority and is never copied into a local repository. Every file explicitly assigned to this root uses the global four-byte `layout_id = 1` header. A required empty family is a header-only file. Equal physical indexes in this root and another authority root have no relationship.

Every `repository_index` below is a direct dense index into this root's `repository.bin` and is that Repository's sole primary key. Index zero is valid. Repository records are append-only: an assigned index is never renumbered, removed, reused, or transferred to another Repository. A `*_index_plus1` uses the global absent encoding. This root never persists a numeric index into repository workflow, Worker Job, or shared-content authority.

The first four Repository indexes are fixed:

```
0 ait-core
1 ait-server
2 ait-python
3 ait-node
```

Every later Repository appends at the next record ordinal starting from four. All server routing, configuration, and operational relationships identify a Repository by `repository_index`, never by Repository name.

The active server-global fixed-family inventory is exactly `repository.bin`. Its active typed-payload inventory is exactly `repository_payload.bin`, and its only active index is `repository_namespace.idx`. `operational_public_sequence.bin`, every Worker Job fixed file, and every Worker Job index are forbidden in this global root. Worker Job files are required in each numeric server Repository authority as declared below. Any other operational `.bin`, payload, or `.idx` file fails activation.

Operational timestamps are non-negative `u64` Unix seconds. A required timestamp is non-zero. An optional timestamp is zero when absent and otherwise non-zero. Admitted RFC 3339 timestamps are normalized to UTC; for a non-negative value, any fractional second is deliberately discarded before the whole-second value is range-checked and stored. A pre-Unix-epoch value, whole-second value greater than `u64::MAX`, or unparseable text fails closed. Repository and Job `updated_at_s` are not earlier than their `created_at_s`. When a Job is locked, `created_at_s <= locked_at_s <= updated_at_s`.

Repository Registry Records

The registry is the installation-scoped authority for Repository identity and routing metadata. It is independent of every repository-scoped workflow and content root.

```
OPERATIONAL_REPOSITORY_RECORD_SIZE = 33

OperationalRepositoryRecord - repository.bin:
u8  repository_meta
u8  lifecycle_kind
u8  namespace_ascii[2]
u8  policy_flags
u32 payload_len
u64 payload_offset
u64 created_at_s
u64 updated_at_s
```

The Repository primary key is the record's physical `repository_index`; it is not duplicated inside the record or payload. Repository names are non-empty immutable UTF-8 display metadata and may repeat without limit. A name is never accepted as identity or routing authority. A non-empty `namespace_ascii` value is unique among `active` and `retiring` records; empty is valid and has no namespace-index row. Historical `purged` identities may share a namespace with a later live Repository, so namespace lookup may return candidates and must verify lifecycle plus the exact two fixed bytes. `repo_name` and `created_at_s` are immutable after creation. Policy, namespace, lifecycle, and `updated_at_s` mutate only by complete-record replacement under the registry lock.

An activated root has at least four Repository records. Records zero through three have exact payload names `ait-core`, `ait-server`, `ait-python`, and `ait-node` respectively and are never tombstoned; retirement uses the retained `lifecycle_kind = purged` record instead. A later record may repeat any of those names without acquiring the reserved record's identity.

The logical Repository default Line is always exact UTF-8 `main`. It is a schema constant, has no fixed or payload field, and is synthesized at an API boundary that exposes a default-Line value. Native creation cannot select a different default Line.

Repository Authority Directory Routing

The configured server Repository-authority parent uses the canonical unsigned base-10 `repository_index` as each Repository directory basename:

```
<server-repository-authority-parent>/
  0/  # ait-core
  1/  # ait-server
  2/  # ait-python
  3/  # ait-node
  4/
  ...
```

Zero is written exactly as `0`; every other basename has no leading zero, sign, whitespace, suffix, prefix, or alternate numeric spelling. Repository name never appears in an authoritative directory name. The directory is the routing container for that Repository's workflow, content, Plan, and Worker Job authority families. The directory itself adds no `.bin` record; the Worker Job child layouts are declared below.

An active parent contains exactly one real, non-symlink directory for every non-tombstoned Repository record and no textual alias directory. Resolving a directory parses its basename as `u32`, verifies that the corresponding `repository.bin` record exists and is not tombstoned, and never scans by Repository name. Compaction preserves every Repository directory basename because it preserves every `repository_index`.

Repository-Scoped Worker Job Records

Each canonical numeric server Repository authority contains exactly one `worker_job.bin` operational queue authority. A required empty file is header-only. This file is Remote Binary input for that one Repository: it is not copied into a local Repository, is not the excluded generic `job.bin`, is not Patchset compact-CI evidence, and is not a queue projection. Active v0 defines no Worker Job payload file.

```
SERVER_WORKER_JOB_RECORD_SIZE = 52

ServerWorkerJobRecord - worker_job.bin:
u8  job_meta
u8  job_kind
u8  state_kind
u8  outcome_kind
u16 attempt_count
u16 max_attempts
u16 error_kind
u16 reserved0
u32 patchset_index_plus1
u32 snapshot_index_plus1
u64 available_at_s
u64 locked_at_s
u64 created_at_s
u64 updated_at_s
```

The owning Repository is implicit in the canonical numeric authority directory. Neither Repository ID nor `repository_index` is duplicated in the fixed record. The direct physical `worker_job_index` record ordinal is the sole Worker Job primary key inside that Repository. Records are append-only: an assigned index is never renumbered, removed, reused, or transferred. A tombstone retains its exact slot.

The complete installation identity is `(repository_index, worker_job_index)`. A bare `worker_job_index` has no meaning outside its routed Repository authority, and v0 stores no separate public or global Job ID. `attempt_count <= max_attempts`; both must fit `u16`, and `max_attempts` is non-zero. `job_kind` is fixed authority; an API `job_type` string is synthesized from it and is never persisted as Job bytes. `available_at_s`, `created_at_s`, and `updated_at_s` are required. Only `running` has a non-zero `locked_at_s`; every other state has `locked_at_s = 0`.

The two fixed domain-reference fields have the exact Job-kind interpretation below. `+1` retains zero for absence. Every non-zero reference resolves inside the same numeric Repository authority as the Job:

job_kind	**API** job_type	patchset_index_plus1	snapshot_index_plus1
2	content.gc	zero	zero
3	content.optimize	zero	zero
4	content.pack	zero	zero
5	land.process	owning Patchset	zero
6	main-seed.refresh	owning Patchset	prior Snapshot or zero
7	patchset.ci	owning Patchset	zero

job_kind	**API** job_type	patchset_index_plus1	snapshot_index_plus1
8	patchset.ci.aggregate	owning Patchset	zero
9	policy.evaluate	owning Patchset	zero
10	reconcile.repo	zero	zero
11	repo.ci	zero	selected Snapshot

`job_kind = 1` is unassigned in active v0. `agent.turn.submit` requires an opaque turn request that is not represented by existing Repository domain authority, so it is not a durable Worker Job until a typed agent-turn schema exists. Every other `job_kind`, non-zero field not assigned by this table, wrong-family target, cross-Repository target, or tombstoned target fails closed. `land.process` derives its Change and revision Snapshot from the resolved Patchset. The Job is committed before any Land exists; execution creates the Land against logical `main` with the fixed server-default `direct` mode. Patchset-related Change and Snapshot data otherwise derive from the resolved Patchset closure. `main-seed.refresh` uses logical `main` and freezes only an independently required prior Snapshot. Native `repo.ci` freezes its selected Snapshot before committing the Job and also operates on logical `main`; neither kind persists a Line index or defers its Snapshot identity to a mutable Repository head.

No variable Worker Job request or input bytes exist. `job_kind` and the two fixed references are the complete durable execution selector. A writer may commit a Job only after every execution-affecting selection has been committed to those fields or to the referenced existing domain authority. Trigger labels, transport labels, scheduler resource keys, priority, retry

delay, runner context, and materialized runtime payload are reconstructed runtime data and cannot alter that durable selection. Content-maintenance and Repository-reconciliation kinds invoke their one server-owned kind-level operation with no per-Job override. Patchset CI and aggregation derive their suite closure from the selected Patchset and its validated Policy/CI authority. Repository CI operates on the selected Snapshot and logical `main` using the server-owned Repository CI contract. If a requested operation cannot be reconstructed exactly under these rules, enqueue fails before allocating a `worker_job_index`.

Operational Repository Payload File

The server-global root and every server Repository authority have no shared string pool and no `operational_payload.bin`. The Repository registry locator addresses the global `repository_payload.bin`. Worker Jobs have no payload locator or payload file.

```
        OperationalRepositoryPayload - repository_payload.bin:
        u16 repo_name_len
        u8  repo_name_bytes[repo_name_len]
```

Repository `payload_len` is non-zero and `payload_offset` is at or after `BIN_HEADER_SIZE`; it is exactly `2 + repo_name_len`, with no padding or trailing field. Repository names are exact valid UTF-8. The complete Repository payload is at most 65,537 bytes.

Rebuildable Registry And Worker Job Indexes

All indexes in this subsection are optional rebuildable accelerators. A file uses exactly the authority root and target family implied by its filename. `repository_namespace.idx` belongs to the server-global registry. The two Worker Job indexes belong separately to every numeric server Repository authority. Persistent rows sort by all fields in declaration order.

```
        OPERATIONAL_NAMESPACE_INDEX_RECORD_SIZE = 8

        OperationalNamespaceIndexRecord - repository_namespace.idx:
        u8  namespace_ascii[2]
        u16 reserved0
        u32 repository_index_plus1
```

Empty namespace bytes `[0x00, 0x00]` have no index row. Non-empty exact-byte namespace candidates may include more than one historical `purged` identity but at most one `active` or `retiring` identity.

```
        SERVER_WORKER_READY_INDEX_RECORD_SIZE = 12

        ServerWorkerReadyIndexRecord - worker_ready.idx:
        u64 available_at_s
        u32 worker_job_index_plus1

        SERVER_WORKER_STATE_INDEX_RECORD_SIZE = 8

        ServerWorkerStateIndexRecord - worker_state.idx:
        u8  state_kind
        u8  reserved0
        u16 reserved1
        u32 worker_job_index_plus1
```

`worker_ready.idx` contains live `queued` rows only. Queue claiming verifies the local Job index, authoritative state, exact `available_at_s`, attempt budget, and absence of a live runtime lease after lookup. `worker_state.idx` contains every live Job. In both files, `worker_job_index_plus1` resolves only inside the same Repository authority that owns the index.

No `.idx` row is written for a tombstoned record. Missing, stale, duplicated, or corrupt indexes are discarded and rebuilt from that authority's `worker_job.bin`. Indexes never repair or override the fixed record.

An installation-wide scheduler enumerates `active` and `retiring` Repository indexes from the registry, reads exact-verified candidates from each local `worker_ready.idx`, and merges them in memory by `(available_at_s, repository_index, worker_job_index)`. That in-memory merge is disposable and is never persisted as a global Job index, sequence, or identity authority.

Operational Metadata And State Encodings

Except where overridden below, a mutable operational `*_meta` byte uses:

```
        bits 0..6 reserved = 0
        bit 7 tombstoned
```

A tombstoned record is retained history, is omitted from indexes, and cannot be the target of a live relationship. Source conversion writes no tombstones.

Repository metadata is:

```
        repository_meta:
        bits 0..6 reserved = 0
        bit 7 tombstoned

        lifecycle_kind:
        1 active
        2 retiring
        3 purged

        namespace_ascii[2]:
        [0x00, 0x00] empty
        [x,    0x00] one-byte namespace
        [x,    y   ] two-byte namespace

        policy_flags:
        bit 0 require_attestation
        bit 1 require_tests
        bit 2 require_lint
        bit 3 require_security_scan
        bit 4 require_license_scan
        bit 5 require_ai_provenance
        bit 6 require_code_review_summary
        bit 7 docs_only_relaxed_checks
```

Each `x` or `y` is one non-zero ASCII alphanumeric, _, or `-` byte. `[0x00, y]`, an embedded zero, a control/non-ASCII byte, and an input longer than two bytes are invalid. Case and bytes are exact; readers perform no trimming, folding, or Unicode normalization. The logical namespace length is derived from the trailing zero and is never stored.

Bits 0 through 6 are the Repository's default requirements. When bit 7 is set and the effective content class is `docs_only`, tests, lint, security scan, and license scan are all not required; attestation, AI provenance, and code-review summary retain their default bits. The logical policy ID and version are fixed to `prototype` and `1` and are not persisted. An API that exposes Repository policy JSON synthesizes the canonical logical object from `policy_flags`.

Every non-tombstoned Repository requires a non-empty name, one valid `namespace_ascii[2]` value, and one `policy_flags` byte. Duplicate names are valid. `active -> retiring -> purged` and `retiring -> active` are the only lifecycle transitions. A `purged` Repository has no live Worker Jobs and cannot be the owner of a new Job. Purged identity, Repository payload bytes, and retained Repository-local Job history remain exact until offline compaction; they are never reconstructed from another domain.

Worker Job state is:

```
        state_kind:
        1 queued
        2 running
        3 succeeded
        4 failed

        outcome_kind:
        0 none
        1 completed
        2 skipped
        3 attached
        4 superseded
        5 failed

        error_kind:
        0 none
        1 retryable_execution
        2 terminal_execution
        3 lease_expired
```

Every live Job is stored under the numeric authority directory of an `active` or `retiring` Repository. `queued` and `running` require `outcome_kind = none`; `succeeded` requires `completed`, `skipped`, `attached`, or `superseded`; and

`failed` requires `outcome_kind = failed`. A new queued Job has `error_kind = none`. A queued retry or running retry may retain `retryable_execution` or `lease_expired`. `succeeded` requires `error_kind = none`; `failed` requires `terminal_execution` or `lease_expired`.

`attached` and `superseded` preserve only why this Job performed no independent successful work. They do not identify another Job. Any active-equivalent or later-successful Job identity used while deduplicating is runtime/projection data and is not Worker Job authority.

Lease credentials are not Binary DB authority. On claim the server creates a cryptographically random 16-byte `lease_token`, keeps the live entry in memory, and mirrors it to a crash-recovery temporary Binary outside every activated global or Repository authority root. The runtime entry is keyed by `(repository_index, worker_job_index, attempt_count)` and carries the token, latest heartbeat time, expiry time, and torn-write detection. It is valid only while the fixed Job is `running`, its `attempt_count` matches exactly, the presented token exact-matches, and the runtime expiry has not passed.

The temporary Binary is a disposable runtime replica, not a layout-1 file family. It is excluded from authority manifests, conversion output, Remote Binary transfer, offline compaction, and domain recovery. It cannot make a queued Job running, repair `worker_job.bin`, or supply Job identity. A missing, corrupt, stale, mismatched, or expired entry invalidates the lease; under the Repository queue lock the server requeues or terminally fails the Job according to its attempt budget, clears `locked_at_s`, and records `lease_expired`. Claim returns the token only after the running Job record and runtime mirror are durable. Heartbeat, completion, and failure require the same token while holding the queue lock. Worker identity is diagnostic runtime data and is not stored in either Worker Job authority or the lease token.

Every numeric state, outcome, or error value not assigned here is reserved and fails closed. Full successful result objects belong to the domain records they mutate. Full failure messages belong to the excluded tracing/log or diagnostic projection boundary. Neither is Worker Job authority.

Operational Mutation And Recovery Boundary

These global and Repository-scoped operational families add no authoritative WAL, journal, transaction ID, or generic generation record. Lock files, temporary replacement files, and fsync bookkeeping are operational filesystem metadata and are not `.bin` authority. If one mutation needs multiple domain locks, writers acquire the server-global registry lock first, followed by Repository-local queue locks in ascending `repository_index`, and release them in reverse:

```
<server-global-registry-root>/01-registry.lock
<server-repository-authority-parent>/<repository_index>/worker-queue.lock
```

Lock acquisition rejects symlinks and a lock owned by another activated root. A Repository lifecycle transition that inspects or tombstones Jobs acquires the registry lock and its own Repository queue lock in the declared order. A Job mutation acquires only its owning Repository's queue lock. That local lock orders the Worker Job fixed family, the non-authoritative runtime lease replica, both local Job indexes, and each Patchset `ci_worker_job_index_plus1` or compact-CI replacement. Readers hold the corresponding shared lock while resolving fixed records; writers hold it exclusively through the domain commit point.

For append creation, the complete fixed detail record is appended and fsynced. The fixed record is the commit point. Rebuildable indexes are written last. The committed record's physical ordinal becomes its permanent `worker_job_index`; no separate allocator, sequence head, reservation record, payload append, or reusable uncommitted identity exists.

An overwrite uses the existing v0 transaction-layer contract: preserve the complete before-image in non-authoritative recovery storage, write one complete fixed record with reserved bytes zero, fsync it, then discard the before-image. Recovery restores the full before-image unless the replacement reached its domain commit point. Fields inside one fixed record are never independently observable.

Additional domain commit rules are:

- Repository creation stages and fsyncs the real numeric authority directory at the next never-used `repository_index`, then appends the Repository record and commits at `repository.bin`. An interrupted pre-commit directory is unreachable staging and is never activated by name. Policy flags, namespace bytes, lifecycle, and update-time changes commit by replacing the complete Repository record.
- Job creation derives the next `worker_job_index` from the complete committed record count and commits at `worker_job.bin` inside the owning numeric Repository authority. A non-`patchset.ci` Job is complete at that point. For `patchset.ci`, the same queue-locked mutation then replaces the complete owning Patchset record to select the new `worker_job_index`; that Patchset replacement is the relationship commit point. Interruption before it may leave a

valid unselected Job, which cannot supply compact CI evidence.

- A claim increments `attempt_count`, creates the new runtime lease entry, writes `state_kind = running` and `locked_at_s`, and commits authority by replacing the complete Job record. A pre-commit runtime entry is ignored unless the fixed state and attempt count match it exactly. A committed running Job without its valid runtime entry is lease-lost and is requeued or failed under the attempt rule. The token is returned only after both writes are fsynced. Heartbeat exact-compares the token and attempt count, persists the later runtime heartbeat/expiry, and replaces the complete fixed Job record with the same later `locked_at_s`. Failure invalidates the runtime entry, clears `locked_at_s`, records the fixed error kind, and either requeues or writes the terminal failed outcome.
- A successful execution commits its result to the existing owning domain authority before replacing the Job with `state_kind = succeeded` and its fixed outcome and `locked_at_s = 0`, then invalidates the runtime lease entry. Selected `patchset.ci` compact evidence is therefore committed by complete Patchset replacement while that Job remains selected before the Job success marker is written. An interrupted domain-first completion is retried idempotently. `attached` and `superseded` commit only that terminal outcome; no second Job identity is persisted. A stale runtime entry cannot revive a terminal Job. Queue indexes and scheduling views are repairable.
- A Repository transition to `purged` first clears every Patchset `ci_worker_job_index_plus1` by complete-record replacement while preserving compact CI evidence, then tombstones every live owned Job, and finally commits the complete Repository lifecycle replacement. Interruption before that final replacement restores both Patchset and Job before-images.

Recovery first validates headers, exact record divisibility, reserved zeros, the four fixed Repository slots, Repository UTF-8, namespace bytes, policy flags, timestamps, outcomes, errors, and all fixed-reference invariants. It exact-validates the numeric Repository-directory closure before opening every repository-scoped Worker Job authority. It then builds the full source-index graph and rejects any live reference to an absent, tombstoned, or wrong-owner record; conflicting live non-empty namespaces; or a non-zero Patchset Job locator that does not exact-verify a same-root `job_kind = 7` Job whose `patchset_index_plus1` selects that Patchset. Duplicate Repository names and equal local Worker Job indexes in different Repository roots are never recovery conflicts.

Only after authority validates may recovery rebuild the registry namespace index and each Repository's local Job indexes. Mutable Repository metadata, Patchset selection, and Job state are never inferred from a later row or an index. Recovery then loads the runtime lease replica; an absent replica is an empty replica. Each entry must exact-match one running Job and its attempt count and must be unexpired; otherwise the entry is discarded and the Job follows the lease-lost rule. An incomplete trailing fixed record is corruption unless the transaction layer proves it is the sole interrupted append. Interior malformed bytes never receive truncation repair.

Operational Capacity And Compaction

Every fixed family and rebuildable index is limited to `u32::MAX - 1` live target indexes because non-zero `*_index_plus1` must remain representable. That fixed-family limit applies independently to `worker_job.bin` in each Repository; no persisted global Job family imposes an additional installation-wide Job-count limit. Because Worker Job tombstones retain their PK slots, its capacity check uses total committed record count rather than live Job count. Direct `u32` source values and counts are preflighted before writing. Repository payload offsets and file sizes must fit `u64`; every `offset + length` calculation uses checked arithmetic.

Offline compaction writes a new inactive registry root and Repository generation, preserving every exact `repository_index` and `worker_job_index`, Repository names, namespace bytes, policy flags, state, kind, fixed references, outcomes, errors, and times. It never reorders, removes, or renumbers a `repository.bin` or `worker_job.bin` record; both Repository and Worker Job tombstones retain their slots and no later identity may reuse them. Other physical families may be densely renumbered only after rewriting their complete reference and index closure. Compaction rebuilds the two local Job indexes without changing Job record order or Patchset locators. It does not read, copy, or create the temporary runtime lease replica and validates the complete target before atomic activation.

Changing a fixed width, state assignment, payload grammar, file family, primary-key rule, conversion source boundary, or capacity requires a new layout conversion and compatible readers and writers. An unchanged `layout_id` does not permit the change.

BINARY DB V0 SCHEMA

Binary DB v0: Payloads, Indexes, and Encodings

Expand every typed payload family, rebuildable index record, metadata bit assignment, enum, and reserved-value rule.

AUDIENCE Codec, validator, and migration implementers

https://ait-native.dev/technical/v1.1.1/binary-db/physical/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

Fixed records frequently point into typed payload files and use compact metadata fields. This chapter gives the complete byte layouts, lookup records, bit assignments, enums, and validation rules needed to decode those values without inventing a generic string or payload format.

Typed Payload Files

The format defines no shared `strings.bin`, `strings.idx`, or `strings.hash`. Variable-length bytes are owned by a typed domain payload file.

```
TaskPayload - task_payload.bin:
u16 title_len
u8  title_bytes[title_len]
u8  intent_bytes[payload_len - 2 - title_len]

ChangePayload - change_payload.bin:
u8  title_bytes[payload_len]

PatchsetSummaryPayload - patchset_summary_payload.bin:
u8  summary_bytes[summary_len]

SnapshotPayload - snapshot_payload.bin:
u16 message_len
u8  message_bytes[message_len]
u8  line_name_bytes[payload_len - 2 - message_len]

TagPayload - tag_payload.bin:
u16 tag_name_len
u8  tag_name_bytes[tag_name_len]
u8  message_bytes[payload_len - 2 - tag_name_len]
```

```
SnapshotLinkPayload - snapshot_link_payload.bin:
u16 worktree_name_len
u16 line_name_len
u16 task_id_len
u16 change_id_len
u16 author_mode_len
u8  worktree_name_bytes[worktree_name_len]
u8  line_name_bytes[line_name_len]
u8  task_id_bytes[task_id_len]
u8  change_id_bytes[change_id_len]
u8  author_mode_bytes[author_mode_len]
u8  model_name_bytes[
      payload_len - 10 - worktree_name_len - line_name_len - task_id_len
      - change_id_len - author_mode_len
    ]
```

```
ActorPayload - actor_payload.bin:
u8  user_name_len
u8  user_id_len
u8  email_len
u8  user_name_bytes[user_name_len]
u8  user_id_bytes[user_id_len]
u8  email_bytes[email_len]
u8  memo_bytes[payload_len - 3 - user_name_len - user_id_len - email_len]

ReviewPayload - review_payload.bin:
u8  message_bytes[payload_len]

WaiverPayload - waiver_payload.bin:
u8  reason_bytes[payload_len]

PlanPayload - plan_payload.bin:
u8  title_bytes[payload_len]
```

```
PlanRevisionPayload - plan_revision_payload.bin:
u16 title_snapshot_len
u16 summary_len
u16 artifact_path_len
u16 artifact_selector_len
u16 artifact_heading_len
u8  title_snapshot_bytes[title_snapshot_len]
u8  summary_bytes[summary_len]
u8  artifact_path_bytes[artifact_path_len]
u8  artifact_selector_bytes[artifact_selector_len]
u8  artifact_heading_bytes[artifact_heading_len]
u8  artifact_blob_id_bytes[
      payload_len - 10 - title_snapshot_len - summary_len - artifact_path_len
      - artifact_selector_len - artifact_heading_len
    ]
```

```
PlanItemPayload - plan_item_payload.bin:
u16 plan_item_ref_len
u16 text_len
u8  plan_item_ref_bytes[plan_item_ref_len]
u8  text_bytes[text_len]
u8  heading_path_bytes[payload_len - 4 - plan_item_ref_len - text_len]
```

```
GitRepositoryPayload - git_repository_payload.bin:
u8  identity_bytes[identity_len]

GitIdentityPayload - git_identity_payload.bin:
u32 raw_len
u32 name_len
u8  raw_identity_bytes[raw_len]
u8  name_bytes[name_len]
u8  email_bytes[payload_len - 8 - raw_len - name_len]

GitCommitMappingPayload - git_commit_mapping_payload.bin:
u32 message_len
u8  message_bytes[message_len]
u8  raw_commit_bytes[payload_len - 4 - message_len]

GitFileMappingPayload - git_file_mapping_payload.bin:
u8  path_bytes[path_len]

GitRefMappingPayload - git_ref_mapping_payload.bin:
u8  ref_name_bytes[ref_name_len]

GitTagMappingPayload - git_tag_mapping_payload.bin:
u32 message_len
u8  message_bytes[message_len]
u8  raw_tag_bytes[payload_len - 4 - message_len]
```

`heading_path_bytes` requires a stable schema-defined encoding. Layout v0 does not choose a new canonical representation. A writer must therefore preserve an already admitted layout-1 encoding or fail closed.

Single-field payloads do not repeat an inner length. Their length comes from the owning record's `payload_len` or, for Patchset summary only, `summary_len`. Multi-field payloads store lengths for every variable field except the last; the last length is derived from the owning record's `payload_len`. Underflow, empty required textual fields, and invalid UTF-8 in

fields declared as UTF-8 are rejected. Fields explicitly declared as opaque Git bytes are not UTF-8 decoded.

Actor, Task, Change, Snapshot, Snapshot Link, Tag, Review, Waiver, and Plan-family records use `u16 payload_len`, limiting one payload to 65535 bytes. Patchset summary uses `u16 summary_len`, requires `1..65535` bytes, and has the same upper bound without becoming a generic Patchset payload. Actor component lengths and Tree path-segment lengths use `u8`, limiting each to 255 bytes. Git identity, commit, and Tag-mapping payloads use `u32 payload_len`; Git repository identity, file path, and ref-name lengths are also `u32`. A value that exceeds `u32` is rejected before writing. `line_name_payload.bin`, `tree_name_payload.bin`, and optional `snapshot_path_payload.bin` store their owning record's normalized UTF-8 bytes.

AIT Tag name/message, Git repository identity, Git identity components, Git file path, and Git ref name must be valid UTF-8 under the current command contract. Git commit message, raw commit, Git Tag message, and raw annotated-Tag bytes are length-bounded opaque bytes and are never Base64-encoded in Binary DB. An identity authored by AIT may use `raw_len = 0`; an identity preserved from Git must retain its raw bytes. Lightweight Git Tags have empty raw Tag bytes.

Attestation, Policy Decision, and Land records own no payload file. Patchset owns only `patchset_summary_payload.bin`; `patchset_payload.bin` is not a v0 file.

Rebuildable Indexes

`.idx` files are lookup accelerators, never identity or authority. They may be absent or rebuilt. A hit is accepted only after comparing the referenced authoritative record or payload bytes. Persistent rows are sorted by lookup-key bytes and then target index.

```
SNAPSHOT_ID_INDEX_RECORD_SIZE = 12
SnapshotIdIndexRecord - snapshot_id.idx:
u64 snapshot_hash48
u32 content_snapshot_index_plus1

SNAPSHOT_PARENT_CHILD_INDEX_RECORD_SIZE = 8
SnapshotParentChildIndexRecord - snapshot_parent_child.idx:
u32 child_snapshot_index
u32 parent_edge_index_plus1

TREE_ID_INDEX_RECORD_SIZE = 14
TreeIdIndexRecord - tree_id.idx:
u8  tree_hash80[10]
u32 tree_index_plus1

TREE_PACK_ID_INDEX_RECORD_SIZE = 12
TreePackIdIndexRecord - tree_pack_id.idx:
u64 pack_hash48
u32 tree_pack_index_plus1
```

```
BLOB_ID_INDEX_RECORD_SIZE = 14
BlobIdIndexRecord - blob_id.idx:
u8  blob_hash80[10]
u32 blob_index_plus1

OBJECT_PACK_ID_INDEX_RECORD_SIZE = 12
ObjectPackIdIndexRecord - object_pack_id.idx:
u64 pack_hash48
u32 object_pack_index_plus1

MANIFEST_HASH_INDEX_RECORD_SIZE = 36
ManifestHashIndexRecord - manifest_hash.idx:
u8  manifest_hash[32]
u32 content_snapshot_index_plus1
```

```
ACTOR_LOOKUP_INDEX_RECORD_SIZE = 12
ActorLookupIndexRecord - actor_lookup.idx:
u64 actor_key_hash
u32 actor_index_plus1

LINE_NAME_INDEX_RECORD_SIZE = 12
LineNameIndexRecord - line_name.idx:
u64 line_name_hash64
u32 line_index

TAG_NAME_INDEX_RECORD_SIZE = 12
TagNameIndexRecord - tag_name.idx:
u64 tag_name_hash64
u32 tag_index
```

```
GIT_REPOSITORY_FINGERPRINT_INDEX_RECORD_SIZE = 16
GitRepositoryFingerprintIndexRecord - git_repository_fingerprint.idx:
u8  fingerprint96[12]
u32 repository_index_plus1

GIT_GENERATION_ID_INDEX_RECORD_SIZE = 12
GitGenerationIdIndexRecord - git_generation_id.idx:
u8  generation_hash64[8]
u32 generation_index_plus1

GIT_COMMIT_OBJECT_INDEX_RECORD_SIZE = 28
GitCommitObjectIndexRecord - git_commit_object.idx:
u32 generation_index
u8  git_object_id[20]
u32 commit_mapping_index_plus1

GIT_COMMIT_SNAPSHOT_INDEX_RECORD_SIZE = 8
GitCommitSnapshotIndexRecord - git_commit_snapshot.idx:
u32 snapshot_index
u32 commit_mapping_index_plus1
```

```
GIT_REF_LOOKUP_INDEX_RECORD_SIZE = 16
GitRefLookupIndexRecord - git_ref_lookup.idx:
u32 repository_index
u64 ref_name_hash64
u32 ref_mapping_index_plus1

GIT_TAG_REF_INDEX_RECORD_SIZE = 8
GitTagRefIndexRecord - git_tag_ref.idx:
u32 git_ref_mapping_index
u32 git_tag_mapping_index_plus1

GIT_OPERATION_ID_INDEX_RECORD_SIZE = 12
GitOperationIdIndexRecord - git_operation_id.idx:
u8  operation_hash64[8]
u32 checkpoint_index_plus1
```

Snapshot parent-child keys admit duplicate candidates and return edge indexes in ascending order; authoritative edge ordinals are verified after lookup. Git ref lookup uses the import source repository index or export target repository index. Hash hits for Tag and Git ref names are accepted only after exact typed payload comparison. Commit-by-Snapshot lookup may return both import and export candidates; generation direction, mapping flags, object format, and recorded order select the landed preference of unchanged import before deterministic export.

Plan Item refs are resolved through a revision's item range and exact payload comparison. Layout v0 defines no persistent Plan Item ref hash index.

Metadata Encodings

All bits and values not assigned below are reserved and must be written as zero.

```
TaskRecord.task_meta:
bit 0 planned
bit 1 snapshotted
bit 2 review_pending
bit 3 validation_pending
bit 4 blocked
bit 5 ready_to_land
bit 6 completed
bit 7 canceled

LocalTaskRecord.local_meta:
bit 0 published
bit 1 abandoned
bit 2 later_promotion_excluded

Remote task/change/mirror remote_meta:
bit 0 tombstone
bit 1 partial
bit 2 conflict
bit 3 stale
```

```
ChangeRecord.change_meta:
bits 0..1 lifecycle: 00 draft, 01 active, 10 landed, 11 archived
bit 2 has_patchsets
bit 3 review_pending
bit 4 validation_pending
bit 5 ready_to_land
bit 6 blocked
bit 7 superseded; valid only with lifecycle 11

ChangeRecord.change_state:
bit 0 canceled; valid only with lifecycle 11
bits 1..7 reserved = 0

LocalChangeRecord.local_meta:
bit 0 published

WorktreeCursorRecord.cursor_meta:
bit 0 has_selected_change
bit 1 has_pending_pre_land_target_snapshot
bit 2 has_pending_landed_snapshot
```

For a Local Change, `local_meta` bit 0 and `published_remote_change_ordinal_plus1` obey the presence and owning-Task invariants in the Change Records section. The four-byte publication slot never stores a global Remote Change index.

```
LineRecord.line_meta:
bit 0 archived
bit 1 tombstone

TagRecord.tag_meta:
bit 7 tombstone

StashRecord.stash_meta:
bit 0 workspace_cleared
bit 7 tombstone

LandRecord.land_meta:
bits 0..2 status_kind:
  000 queued
  001 running
  010 succeeded
  011 blocked
  100 failed
  101 canceled
  110 updating
  111 reserved
bit 3 has_pre_land_target_snapshot
bit 4 has_landed_snapshot
bits 5..6 mode_kind:
  00 direct
  01 merge
  10 ff_only
  11 reserved
bit 7 tombstone

LandRecord.failure_kind:
0 none
1 base_stale
2 policy_blocked
3 review_blocked
4 ci_blocked
5 conflict
6 target_update_failed
7 internal_error
```

```
ContentSnapshotRecord.snapshot_meta:
bits 0..1 snapshot_kind: 00 line, 01 stash
bit 2 has_message
bit 3 has_line_name_payload
bit 4 parent_edges_authority
bit 5 has_root_locator
bit 7 tombstone

ContentSnapshotRecord.history_flags:
bit 0 remote_head_history_boundary
bits 1..7 reserved = 0

SnapshotLinkRecord.link_meta:
bit 0 has_change
bit 1 has_session
bit 2 has_checkpoint
bit 3 has_worktree_name
bit 4 has_line_name
bit 5 has_author_or_model
bit 7 tombstone
```

```
GitRepositoryRecord.repository_meta bits 0..1:
00 Git import source; public prefix GSR
01 Git export target; public prefix GTR
10 AIT export source; public prefix ASR
11 reserved

GitRepositoryRecord.object_format_kind:
0 not_applicable for an AIT repository
1 sha1

GitGenerationRecord.generation_meta bits 0..1:
00 import; public prefix GIT-IMP
01 export; public prefix GIT-EXP
10..11 reserved

GitIdentityRecord.identity_meta:
bit 0 has_raw_identity

GitCommitMappingRecord.mapping_meta:
bit 0 signed
bit 1 imported_unchanged
bit 2 deterministic
```

```
GitRefMappingRecord.ref_meta:
bits 0..1 ref_kind: 00 branch, 01 tag, 10 symbolic_ref, 11 reserved
bit 2 has_snapshot
bit 3 has_target
bit 4 has_expected_previous_git_object_id

GitTagMappingRecord.tag_mapping_meta:
bit 0 signed
bit 1 deterministic
bit 2 has_raw_tag

GitOperationCheckpointRecord.checkpoint_meta bits 0..1:
00 running
01 completed
10..11 reserved

Git git_object_type_kind:
0 commit
1 tree
2 blob
3 tag
```

```
PlanRecord.plan_meta:
bits 0..1 plan_state: 00 draft, 01 archived, 10 superseded
bit 2 published
bit 3 stale
bit 4 conflict
bit 5 tombstone

PlanRevisionRecord.revision_meta:
bit 0 published
bit 1 stale
bit 2 conflict
bit 3 has_items
bit 4 tombstone

PlanItemRecord.item_meta:
bits 0..1 checkbox_state: 00 none, 01 open, 10 done
bit 2 has_item_ref
bit 3 taskable
```

```
TreePackRecord.pack_meta:
bit 0 ready
bit 1 corrupt
bit 2 sparse_physical_ordinals
bit 7 tombstone

TreePackRecord.pack_format_kind:
0 zip_deflate_tree_v1
1 zstd_chunked_tree_v1

TreeRecord.tree_meta:
bit 7 tombstone

TreeEntryRecord.entry_meta bits 0..1:
00 blob
01 tree
```

```
ObjectPackRecord.pack_meta:
bit 0 ready
bit 1 corrupt
bit 2 pruned
bit 7 tombstone

ObjectPackRecord.pack_format_kind:
0 zip_deflate_v2
1 zstd_chunked_v1

ObjectPackMemberRecord.member_meta:
bits 0..1 entry_kind: 00 full, 01 delta
bits 2..3 compression_kind: 00 none, 01 deflate, 10 zstd
bit 7 tombstone

BlobRecord.blob_meta:
bit 0 has_pack_member
bit 1 pruned
bit 7 tombstone

BlobRecord.hash_kind:
0 sha256
```

```
PatchsetRecord.patchset_meta:
bit 0 withdrawn
bit 1 invalidated
bits 2..4 author_mode_kind:
  000 human_only
  001 human_with_ai_assist
  010 ai_with_human_review
  011 ai_only_experimental
  100 agent (legacy conversion only)
  101 codex (legacy conversion only)
  110 xhigh (legacy conversion only)
  111 reserved
bits 5..6 publish_state_kind:
  00 published
  01 reserved
  10 superseded
  11 reserved
bit 7 evaluation_pending

AttestationRecord.attest_meta:
bits 0..1 verification_state: 00 unknown, 01 pending, 10 pass, 11 fail
bit 2 revoked
bit 3 require_tests_pass
bit 4 require_human_review
bit 5 require_lint_pass
bit 6 ci_backed
bit 7 tombstone

ActorRecord.actor_meta:
bits 0..2 actor_kind:
  000 user
  001 team
  010 bot
  011 service_account
  100 automation
  101 unknown
bit 3 has_user_id
bit 4 has_email
bit 5 has_memo
bit 7 tombstone
```

Native v0 Patchset writers emit only the four non-legacy author modes. The three legacy codes exist solely to preserve exact admitted source rows during conversion; unknown author text fails closed. Native v0 Patchset writers emit only `published` or `superseded` publish state. A legacy source Patchset state `selected_for_landing` is a non-authoritative projection and normalizes to `published`; it is never written as `01`, used to infer selection, or allowed to override `RemoteChangeRecord.selected_patchset_index_plus1`. If a valid source Change selected pointer disagrees with stale Patchset projections, the Change pointer is authoritative and conversion does not fail solely for that projection disagreement. A missing, invalid, or ambiguous source Change pointer still fails whenever source selection is required. The selected pointer may reference a `published` or `superseded` legacy row only when the source Change pointer explicitly does so and all ownership and liveness rules otherwise pass.

When `evaluation_pending` is set, public Patchset `evaluation_state` is `pending` even if an older Policy Decision exists. When it is clear, a latest live non-pending Policy Decision is required and supplies the exact `pass`, `soft_fail`, `hard_fail`, or `waived` value. New Patchsets and any review, attestation, waiver, or policy-input mutation that invalidates cached policy set this bit as the status-last commit marker; a completed non-pending Policy evaluation appends its decision before clearing the bit.

```
ReviewRecord.review_meta:
bits 0..2 action_kind:
  000 request
  001 comment
  010 approve
  011 request_changes
  100 dismiss
bit 3 blocking
bit 4 task_lane
bit 5 code_review_summary
bit 6 defer
bit 7 tombstone

PolicyDecisionRecord.policy_meta:
bits 0..2 decision_kind:
  000 pending
  001 pass
  010 soft_fail
  011 hard_fail
  100 waived
bit 7 tombstone

WaiverRecord.waiver_meta:
bit 0 revoked
bit 7 tombstone
```

```
PolicyCheckRecord.check_kind:
0 require_attestation
1 ai_provenance
2 code_review_summary
3 tests
4 lint
5 security_scan
6 license_scan
7 required_human_review
8 ci_rollout_phase
9 ci_patchset_suite

PolicyCheckRecord.check_status:
0 absent
1 not_required
2 pending
3 pass
4 hard_fail
5 soft_fail
6 waived
7 optional_fail
```

BINARY DB V0 SCHEMA

Binary DB v0: Identity, Recovery, and Conversion

Expand public handle scopes, atomic write and recovery boundaries, conversion acceptance, capacity, and exclusions.

AUDIENCE Migration, recovery, and compatibility implementers

https://ait-native.dev/technical/v1.1.1/binary-db/recovery-conversion/

1.1.1 documentation route · Binary DB v0 · `layout_id = 1`

The final schema chapter connects physical records to public identity, crash recovery, and admitted conversion. It also states hard capacity limits, explicitly excluded domains and files, and the compatibility boundary for layout v0.

Public Identity And Compact Handles

- Task identity is authority-scoped and derived from dense record ordinal.
- Change uses `C-01..C-64` for ordinal `0..63`.
- A published Local Change's Remote identity is the owning Local Task's published Remote Task index paired with the stored Remote Change ordinal; it is not a global Remote Change index.
- Land uses `L-01..L-64` for ordinal `0..63` within its owning Change; its full public identity includes the Task and Change identity.
- Patchset uses `P-01..P-64` for ordinal `0..63` within its owning Change; its full public identity includes the Task and Change identity.
- Attestation uses `A-01..A-64` for ordinal `0..63`.
- Review uses `R-01..R-64` for ordinal `0..63` within its owning Patchset; its full public identity includes the Task, Change, and Patchset identity.
- Policy Decision uses `K-01..K-64` for ordinal `0..63` within its owning Patchset; its full public identity includes the Task, Change, and Patchset identity.
- Waiver uses `W-01..W-64` for ordinal `0..63`.
- Task Snapshot Link uses `S-01..S-256` for ordinal `0..255`.
- Stash identity is derived from `stash_index`, for example `STH-000001`.
- AIT Tag public identity is its name; stable internal references use the dense `tag_index` and never require a textual Tag ID.
- Canonical Snapshot identity is `SNP-` plus a 48-bit hex suffix.
- Tree and Blob public suffixes are 80 bits.
- Tree Pack and Object Pack public suffixes are 48 bits.
- Git repository fingerprints are 96-bit suffixes, import/export generation and operation identities are 64-bit suffixes, Plan hashes are 96-bit suffixes, and Git mapping identities are derived 80-bit `GIM-*` suffixes.

Every eight-bit workflow handle uses the following bit model within its declared owning scope:

```
        bits 0..5 ordinal 0..63 within (owning identity, kind)
        bit 6 tombstone/deleted
        bit 7 indirect/extended
```

Change, Attestation, and Waiver retain their declared Task-scoped allocation. Patchset and Land allocate within their owning Change. Review and Policy Decision allocate within their owning Patchset. Therefore a Patchset or Land ordinal may repeat across different Changes in one Task, and a Review or Policy ordinal may repeat across different Patchsets.

The complete owning identity disambiguates every repeated numeric suffix.

Layout v0 assigns no extended-handle payload format. A writer that would exceed a declared ordinal range must fail before writing; it must not interpret the indirect bit as permission to invent one.

Multi-File Write And Recovery Boundary

Layout v0 defines no workflow `wal.bin`, `journal.bin`, transaction-ID field, per-record WAL sequence, or Binary DB activation generation token. A `GitGenerationRecord` identifies imported Git source state or a deterministic export plan; it is not a database transaction token. The writer transaction layer owns single-writer/per-task locking, dependency-first writes, ordered fsync, append-only commit points, and repairable head/index projections.

The appended detail record is the domain commit point for Patchset, Review, Policy Decision, Waiver, Attestation, and Land creation. Associated task, change, or patchset head records are repairable projections. Land status mutation uses the status-last protocol: terminal/reset data is written and fsynced before `status_kind` is written and fsynced as the commit marker.

`tree_entry_range.bin` is the sole ordinal-aligned fixed dependency, never a payload or rebuildable index. Creation writes and fsyncs each required range row before its owning Tree commit record. At a stable activation boundary its count equals the Tree owner-file count; recovery may discard only an uncommitted trailing dependency and must fail closed on a missing committed dependency or an interior ordinal mismatch. Task close mutation occurs entirely inside `task.bin`; Change archive/reopen mutation occurs entirely inside `change.bin`. Land creation appends and durably commits one complete `land.bin` record. Each mutation replaces or appends its one complete fixed record. The transaction layer restores the before-image if an overwrite does not reach its commit point; no ordering between fields inside one record is observable.

Recovery validates every header and record alignment, walks committed detail records, repairs `latest_*_index_plus1`, counts and next ordinals, recomputes the latest/current Patchset indexes, validates each Remote Change selected Patchset pointer by ownership and liveness without reconstructing it from Patchset metadata, ignores orphan dependency/payload rows, and rejects duplicate ordinals inside each declared identity scope as corruption. The Task indexes for Patchset, Review, Policy Decision, and Land rebuild physical inventory order only. Their owner indexes rebuild ordinal chains and next-ordinal allocation: Change for Patchset and Land, Patchset for Review and Policy Decision. Equal numeric ordinals in distinct owner scopes are valid.

Snapshot creation appends every validated parent edge and fsyncs the edge file before appending the child Snapshot record with bit 4 set. The Snapshot record is the commit point; an interrupted write may leave only orphan edges, which recovery ignores. Recovery also verifies that a remote-head history boundary has no local parents and that a zero-parent non-boundary Snapshot is a true root. A Tag payload is durable before a Tag record append/update.

Tree creation appends normalized Tree-entry rows and the matching fixed range before the Tree commit record. Recovery validates normalized ranges against the owning Tree and validates every physical pack ordinal according to the Tree Pack's sparse-ordinal bit before admitting the pack projection.

Git repository, generation, identity, parent, file, and typed payload rows are dependencies. The immutable Git commit/ref/Tag mapping record is its mapping commit point and is fsynced before its rebuildable index candidates. Recovery ignores dependency rows not reachable from a committed mapping, validates all mapping ranges and numeric AIT references, then rebuilds Git indexes. A Git operation checkpoint writes its cursors and time before writing and fsyncing `checkpoint_meta`; replay of a completed checkpoint is a no-op reconstructed from immutable mappings.

Bin-to-Bin Conversion Acceptance

A converter must classify every source field as an exact fixed-field/bit mapping, the one admitted Patchset summary payload, a schema-defined derivation, or an explicitly non-authoritative projection. A field outside those classes fails closed; it must not be copied into a generic payload.

An offline converter writes a separate target generation and may assign new dense physical target record indexes. It preserves source Task order and, for every other authoritative file family, preserves validated source record order among emitted rows. Before writing records it builds a complete source-index to target-index map for every admitted family. It then rewrites every owning index, `*_index`, `*_index_plus1`, previous-record link, selected/latest pointer, fixed range start, ordinal-aligned side row, and recalculated payload offset through those maps. Rebuildable `.idx` files are generated from the completed target authority rather than copied.

Physical renumbering never changes `change_ordinal`, `patch_ordinal`, `attest_ordinal`, `review_ordinal`, `policy_ordinal`, `waiver_ordinal`, `land_ordinal`, or `snapshot_ordinal`; their public handles retain the exact source ordinal in each declared scope. Patchset ordinals are preserved within Change, Review ordinals within Patchset, Policy Decision ordinals within Patchset, and Land ordinals within Change. Repetition across different owning Changes or Patchsets is valid and is not renumbered. For Policy Decision, an admitted legacy `.../P-##/POL-##` identity maps the exact `POL-##` suffix to Patchset-scoped `K-##`.

Any duplicate within one declared owner scope, more than 64 rows of one kind in that owner scope, ambiguous ownership, or source ID/fixed-sequence disagreement fails closed. Task Patchset, Review, Policy, and Land inventory links are rebuilt in emitted physical record order. Change Patchset and Land links, and Patchset Review and Policy links, are rebuilt by exact ordinal. Any source reference without exactly one target map entry fails closed. Source files remain immutable, and the fully validated target generation is activated atomically. No old-to-new mapping bin or payload is added.

The only admitted supplied `diff_stats` disagreement is an explicitly selected offline legacy stale-zero gate. The gate uses an exact configured allowlist of complete Patchset identities; partial, ordinal-only, or wildcard matching is forbidden.

For each named Patchset, the supplied value must be the complete object with `files_added = 0`, `files_changed = 0`, `files_deleted = 0`, and `files_modified = 0`, plus exactly empty `paths.added`, `paths.deleted`, and `paths.modified` arrays. Its canonical base and revision Snapshot references must both resolve to present, live v0 Snapshot authority, and exact comparison of their complete Trees must succeed and produce a non-empty difference. The converter then uses that recomputed difference only to validate acceptance; the target stores no diff-stat value.

Without the explicit gate, every disagreement fails closed. Even with the gate, an unlisted or partially matched identity, a missing/extra/renamed `diff_stats` member, a non-zero supplied count, a non-empty supplied path array, a zero recomputed difference, a missing/tombstoned/malformed Snapshot, an unreadable or malformed Tree, or any other disagreement fails closed. No wildcard, ordinal-only match, fallback Snapshot, reconstructed source value, new field, bit, bin, index, payload, record-width change, or `layout_id` change is admitted.

The only admitted alternate source Task status spelling is an explicitly selected offline gate bound to an exact repository key and complete locked source-manifest digest. The server orchestrator verifies both values before forwarding the gate. Under that gate only, source `status = "canceled"` maps to the existing `TaskRecord.task_meta` bit 7, identically to source `status = "abandoned"`. The legacy payload has no Task close-time field, so a converted terminal Task retains `closed_at_s = 0` under the existing unknown historical close-time rule.

Without the explicit gate, source `status = "canceled"` fails closed. Even with it, another manifest, another repository key, a partial or case-folded spelling, or any other unknown Task status fails closed. The gate does not change the conversion report and adds no field, bit, bin, index, payload, record-width change, reconstruction marker, inferred timestamp, or `layout_id` change.

The first reference-omission exception is an explicitly selected offline legacy salvage mode for a Patchset whose canonical `revision_snapshot_id` is absent from the same locked source generation's content Snapshot authority. Without that explicit mode the missing reference fails closed. With it, the converter emits no row for that Patchset and emits no Review, Policy Decision or Check, Attestation, or Land that depends on the omitted Patchset. If the omitted Patchset is the owning Change's authoritative current/latest or selected Patchset, the converter emits no row for that Change or any of its Patchsets and dependents; it never chooses a fallback current or selected Patchset. The owning Task remains present.

This exception does not admit an absent or sentinel Patchset Snapshot index, does not fabricate or tombstone a Snapshot, and does not excuse a missing base Snapshot, a malformed Snapshot ID, a tombstoned or corrupt present Snapshot, or any unrelated missing authority. Target Change, Patchset, Review, Policy, Attestation, Land, and Actor rows retain source order among emitted rows, and every surviving fixed reference, chain head, count, selected/latest pointer, ordinal-aligned side row, and rebuildable index is densely remapped. The conversion report must enumerate the omitted Change and Patchset identities and the emitted/omitted dependent-row counts; omission is never silent and never changes source bytes.

The second omission exception is a separately selected named legacy salvage for one exact configured Change identity. That source Change claims landed but has no surviving successful Land and no complete accepted-Patchset/Snapshot closure from which one can be reconstructed. Without the exact named option, conversion fails closed. With it, the converter omits that Change, every Patchset and dependent row owned by it, and every Land row that references it; it does not fabricate a Land, choose a fallback Patchset, infer a timestamp, or affect any other Change. The report must name the selected Change and all dependent-row counts. No wildcard, status-based, or implicit omission is permitted.

The admitted legacy landed-Change normalization is narrower than the omission exceptions above. A Change with `current_patchset_number = 0` rebuilds latest/current only from its greatest surviving `patch_ordinal` and only when

its explicit selected pointer identifies that same Patchset. Exact Land rows remain authoritative even when the duplicated Change `status` or `landed_at` projection differs. Every attempt is preserved, any successful attempt makes the Change lifecycle landed, and the greatest successful `land_ordinal` supplies landed state even when a later attempt did not succeed. The current mutable selected pointer need not equal any historical Land's accepted Patchset. These rules discard only duplicated projections, never an authoritative row, and add no field, bit, bin, payload, fallback Patchset, or inferred timestamp.

Before conversion, a server orchestrator may copy source authorities into an isolated frozen generation without acquiring or creating source locks. The freeze operation captures every current authority data file before copying, copies those files rather than trusting the live generation's possibly stale file list, captures the complete source inventory again, and requires exact before/after/copy equality by `relative_path`, `byte_size`, and complete SHA-256. It also requires the source registry manifest's exact bytes to remain unchanged across the operation. Any drift fails closed and leaves no published target. The fresh frozen generation uses the existing manifest schema with file lists and fingerprints for exactly those copied bytes. Repository manifest copies and operational lock files are excluded from the data-file list. Conversion accepts that frozen generation only when every declared file matches and no undeclared authority data file exists.

No source lock file is required or created in either the selected source or the frozen copy. The orchestrator may create a missing legacy `.bin` or `.idx` as a header-only file only when the selected source manifest explicitly proves that exact file or its entire declared family has zero records, or when the authority class does not contain that whole optional family and no surviving source record can reference it. A partially present family, a non-zero manifest count, an unresolved reference, or missing proof fails closed. Header synthesis is never performed in the selected source root, and the conversion report enumerates every staged header-only file.

The newly admitted missing-time zero cases are limited to `RemoteTaskRecord.updated_at_s`, `RemoteTaskRecord.fetched_at_s`, `RemoteChangeRecord.fetched_at_s`, an archived `RemoteChangeRecord.archived_at_s`, and `ActorRecord.created_at_s`/`last_seen_at_s`. They apply only when the selected legacy source format demonstrably has no corresponding time field. The previously enumerated legacy Task `closed_at_s` and first successful Land `submitted_at_s` exceptions remain unchanged. No other missing or invalid time becomes zero, and no zero authorizes a synthetic time or reconstruction bit.

Source `identity_source` is an annotation, not stored identity. It may be omitted only when absent or when it agrees with the canonical identity derived from the selected authority root and numeric record relationship. A non-default, conflicting, or unresolvable value fails closed. Source `published_remote_name` is likewise not stored. It may be omitted only when absent/empty or exactly equal to the one remote authority explicitly selected for the conversion; a different or ambiguous remote name fails closed.

A source Snapshot with no parents becomes a true v0 root only when the source authority proves it had no parents. A proven-empty authority therefore remains empty, and two or more independently proven roots remain separate components of one DAG forest. Every surviving non-boundary Snapshot must reach one such root. If source evidence instead shows ancestry whose parents were not imported, the converter sets `remote_head_history_boundary`; missing evidence that cannot distinguish the two cases fails closed. No converter infers a root or history boundary from position, record order, component size, Line membership, or current-head status alone.

Textual public IDs are parsed only through their declared compact ordinal or hash rules and then validated against the owning fixed records. Source cache keys, derived labels/messages, duplicated names, diff statistics, and result projections may be discarded only under the explicit derivation rules in this authority. Unknown enum text, overflow, disagreement, missing authority, or ambiguous resolution aborts conversion before activation.

Capacity And Expansion

Writers preflight representable counts and narrow fields before any partial append:

```
current_count  = (file_size - BIN_HEADER_SIZE) / record_size
projected_count = current_count + records_to_append
```

Overlong `u8`/`u16` text is an input validation failure, not permission to widen layout v0. Except for the explicitly enumerated Task `closed_at_s` tail, Change lifecycle tail, Land target-Line tail, and Patchset summary/Worker-Job-locator/time-width corrections, the enumerated Git mapping time-field removals, and the separately scoped operational families, a genuine record/index capacity expansion requires an explicit migration to a new `layout_id`, converts the complete dependent reference closure, writes separate new files, validates them, and performs an atomic swap. The bounded operational conversions relocate and narrow the Worker Job record, retire its global identity families, add only the two declared local indexes, and append only the declared Patchset locator; the earlier fixed-time correction narrowed only its named

timestamp-bearing records and ready index. The u64-second conversion subsequently widens only the complete enumerated normal time-field set. Neither conversion permits altering another field or file. Mixed record widths in one file are forbidden even though the persisted layout number remains one.

Snapshot parent count is capped at 1,024 even though `parent_count` and `parent_ordinal` are `u16`. AIT Tag payloads are capped by `u16 payload_len`. Git payload and single-field length values are capped by `u32`; Git SHA-1, fingerprint, generation, operation, and Plan-hash byte arrays have the exact fixed widths declared above. Except for the explicitly removed AIT-generated mapping timestamps, narrowing, truncation, hash-prefix substitution, or accepting SHA-256 Git Object IDs in the 20-byte fields is forbidden.

The one remaining ordinal-aligned side-file family preflights its owner count and all `u32` indexes before writing. It is a layout-1 schema extension, not permission to widen an existing record or move its fields into payload.

The Task tail is a one-time layout-1 conversion, not a general widening precedent. Conversion must explicitly select the old local 40-byte or remote 36-byte source layout, rewrite the complete file to the corrected local 44-byte or remote 40-byte layout, and activate only after full validation.

The Change tail is likewise a one-time layout-1 conversion. Conversion of the immediately preceding split representation must explicitly select the 44-byte Change source layout and its aligned 8-byte lifecycle source, rewrite and validate the complete 52-byte `change.bin`, omit `change_lifecycle.bin` from the new generation, and activate only after every base-Line, archive-time, owner, and Snapshot relationship validates.

The Land tail is likewise a one-time layout-1 conversion. Conversion of the immediately preceding split representation must explicitly select the 32-byte Local or 36-byte Server Land source layout and its aligned 4-byte target-Line source, rewrite and validate the complete 36-byte Local or 40-byte Server `land.bin`, omit `land_target_line.bin` from the new generation, and activate only after every target-Line, owner, Patchset where present, Snapshot, ordinal-chain, status, and timestamp relationship validates.

The Patchset summary locator is likewise a one-time layout-1 conversion. Conversion of a 47-byte source must first form and validate the declared 57-byte prefix plus `patchset_summary_payload.bin`. The later Worker-Job-locator correction then explicitly selects that exact 57-byte prefix, appends `ci_worker_job_index_plus1`, writes the complete 61-byte `patchset.bin` in a separate generation, and atomically activates only after every summary range, same-Repository Worker Job target, and Patchset relationship validates.

The repository-local Worker Job identity change is also a one-time layout-1 conversion. Conversion must explicitly select the declared 100-byte source record when present, remove its `u64 job_id`, rewrite and validate the complete 92-byte `worker_job.bin`, replace the superseded global Job sequence/index files with the two declared Repository-local indexes, and activate only after every permanent local index, typed payload, Patchset locator, state, and timestamp validates.

The later Worker Job fixed-record change is another one-time layout-1 conversion. Conversion must explicitly select that exact 92-byte source record and its four typed payload families, resolve and validate their complete closure, write the declared 88-byte `worker_job.bin` plus `worker_job_input_payload.bin` in a separate inactive generation, and omit `worker_job_request_payload.bin`, `worker_job_result_payload.bin`, `worker_job_error_payload.bin`, and `worker_job_lease_owner_payload.bin` from the target. Activation occurs only after every Job kind, fixed reference, normalized input, outcome, related Job, error, lease hash, Patchset locator, state, and timestamp validates.

The no-payload and runtime-lease change is another one-time layout-1 conversion. Conversion must explicitly select that exact 88-byte source record and `worker_job_input_payload.bin`, resolve and validate their complete closure, write the declared 60-byte `worker_job.bin` in a separate inactive generation, and omit `worker_job_input_payload.bin` from the target. It discards the source input ranges and `lease_owner_hash` only after the fixed references, source payload contract, and quiesced no-running-Job gate validate. Conversion writes no runtime lease replica. Activation occurs only after every Job kind, fixed reference, outcome, related Job, error, Patchset locator, state, and timestamp validates.

The later Worker Job direct-reference change is another one-time layout-1 conversion. Conversion explicitly selects that exact 60-byte source record and writes the declared 52-byte `worker_job.bin`. For `land.process`, the old subject Land must resolve to an exact `main`, `direct` same-Repository Server Land and its accepted Patchset becomes `patchset_index_plus1`. For `main-seed.refresh`, the old subject Patchset and auxiliary prior Snapshot become the two named fields; its old context Line must be exact `main` and is discarded. Patchset CI, aggregation, and Policy subjects become `patchset_index_plus1`; a `repo.ci` subject becomes `snapshot_index_plus1` after its old context Line exact-validates as `main`. Content maintenance and Repository reconciliation require all three old domain references to be zero. The old related-Job reference exact-validates the `attached` or `superseded` source outcome when present and is then discarded. Activation occurs only after every new named reference, Job kind, outcome, error, Patchset locator, state, and timestamp validates. This historical rewrite does not make a native v0 enqueue depend on a pre-existing Land.

The fixed-time change is another one-time layout-1 conversion. It selects the exact 33-byte Repository, 52-byte Worker Job, 12-byte ready-index, and 61-byte Patchset predecessors and rewrites the declared 25-byte, 36-byte, eight-byte, and 57-byte targets. Admitted RFC 3339 operational timestamps are normalized and deliberately reduced to whole seconds exactly as declared above. Existing Patchset `ci_completed_at_s` values must fit `u32`; an overflow fails before any target write rather than truncating. The Patchset rewrite preserves its first 28 bytes and logical CI, summary, and Job-locator values while shifting the tail to the corrected offsets.

The same correction selects exact 100-byte commit-mapping, 84-byte ref-mapping, and 52-byte Tag-mapping predecessors and writes their declared 92-byte, 76-byte, and 44-byte forms without `recorded_at_unix_nanos`. Before discarding that AIT-generated value, conversion verifies that physical record order reproduces every source latest-record selection; disagreement fails closed. The signed Git identity timestamp and timezone remain byte-for-byte semantic source data. Activation occurs only after exact record divisibility, every timestamp range and ordering rule, all indexes, and the full dependent reference closure validate in an inactive generation.

The u64-second change is another one-time layout-1 conversion. Its only admitted predecessor selector is exact `u32-time-v0`; selecting `layout_id = 1` alone or guessing from file divisibility is insufficient. The selector denotes the complete immediately preceding active v0 representation whose normal `*_at_s` fields and widths are enumerated in the correction table above. A missing selector, another selector, an undeclared file, a source file that is not exactly aligned to its named predecessor width, or a mixed-width file fails before any target publication.

Conversion freezes or read-locks the complete source authority and records its full file inventory and fingerprints. For every fixed record it copies all bytes before, between, and after the declared time fields in exact declaration order and zero-extends each little-endian `u32` second value to the corresponding little-endian `u64`. It does not parse, round, reorder, infer, or replace a timestamp. It copies every unaffected payload and object byte exactly. `GitIdentityRecord.timestamp_s` is not converted. Worker `worker_ready.idx` is rebuilt from the converted authoritative Job records; all other rebuildable indexes are either rebuilt from converted authority or exact-validated against the unchanged identity and ordinal keys.

The converter writes a complete inactive generation, validates every corrected record with the active v0 codecs, validates the complete cross-file reference and content closure, and compares the still-locked source fingerprint with its captured precondition. Activation requires compatible readers and writers, rechecks that source precondition, and atomically selects only the complete generation while retaining a recoverable previous generation or direct authority. A conversion failure or post-freeze source change leaves the active authority untouched. An active file never mixes the predecessor and corrected record widths, and no runtime may write either representation while conversion or activation holds the authority lock.

Excluded Domains And Files

Layout v0 does not define:

- `wal.bin`, `journal.bin`, or per-domain transaction fields; the declared Git import/export content generation is the only v0 generation-ID exception;
- Local Patchset files, a selected-Patchset cursor in Local Change records, or selected-for-landing authority in immutable Patchset metadata;
- generic `ci.bin` or `job.bin`; operational jobs use only the typed Repository-scoped `worker_job.bin` family, while compact Patchset CI evidence and its selected local Job locator remain in the Patchset record;
- generic `patchset_payload.bin`, `attest_payload.bin`, `policy_payload.bin`, or `land_payload.bin`; the narrowly typed `patchset_summary_payload.bin` is the sole Patchset exception;
- Task close time in any payload or side file; the owning Task record's `closed_at_s` tail field is its sole authority;
- Change base-Line/archive state in any payload or side file; the owning Change record's fixed tail fields are their sole authority;
- Land target-Line identity or normalized Tree-entry ranges in any payload file; the owning Land record tail and fixed Tree range side record declared above are their sole authorities;
- server Stash files;
- a conversion-only Actor file or generic Actor payload; conversion uses the existing `actor.bin` and `actor_payload.bin` authority;
- a shared string pool;

- `object_pack_data.bin`; object bytes remain in their pack container;
- a repository-wide current-Line field in `line.bin`;
- a Snapshot parent-index extension in `snapshot_payload.bin`;
- a global published Remote Change index in `LocalChangeRecord`;
- Git mirror mappings, last-mirrored-head state, mirror direction/phase, operational result JSON, or a generic Git mapping payload;
- Worker Job result JSON, full error text, textual lease-owner identity, or active definitions of `worker_job_request_payload.bin`, `worker_job_result_payload.bin`, `worker_job_error_payload.bin`, and `worker_job_lease_owner_payload.bin`; `worker_job_input_payload.bin`, `input_len`, `input_offset`, and `lease_owner_hash` are also retired from active v0;
- a Worker Job Land index, target-Line index, related-Job index, or second Snapshot-reference field; active v0 stores only the named Patchset and single Snapshot references declared in `worker_job.bin`;
- `identity_source`, `published_remote_name`, duplicated AIT Line/Tag names in Git mappings, or a textual stored Line ID;
- a generic operational payload, server scheduler-policy/configuration payload, metrics/time-series store, tracing/log store, notification/outbox store, package artifact bytes, external-provider token vault, or temporary runtime lease replica as Binary DB authority;
- additional server-global operational domains; active operational v0 contains only the global Repository registry/name payload/namespace index and the Repository-scoped Worker Job families declared above;
- a server-global Worker Job fixed record, payload, public sequence, or scheduling/identity index; fixed Job authority and both rebuildable Job indexes remain inside each numeric Repository authority;
- retired runtime/planning sessions or Test inventory/coverage data as Binary authority; and
- a server-global numeric reference into repository workflow/content/Plan authority. Those cross-root references remain exact typed public-identity bytes as declared above.

Legacy SQLite may be read by explicit one-time import/export tooling, but it is not an authoritative runtime fallback. The landed `.ait/git-interop/v1` JSON mapping/checkpoint store may likewise be read only by an explicit one-time conversion into the declared Git fixed records and typed payloads; it is not a parallel runtime authority. Optional caches and rebuildable indexes must never become the sole source of a domain relationship.

REFERENCE

Distribution and Release

Understand the seven-component compatibility family, install-channel roles, and exact 1.1.1 authority.

AUDIENCE Developers, operators, and packagers

https://ait-native.dev/technical/v1.1.1/distribution/

One compatibility family

ait-native 1.1.1 admits seven components together:

- `ait` - native repository and workflow CLI;
- `ait-agent` - optional transport-worker configuration and supervision CLI;
- `ait-agent-worker` - native Telegram, LINE, Discord, and Slack worker runtime;
- `ait-server` - remote protocol and durable authority;
- `ait-runner` - remote native execution plane;
- `ait-python` - direct in-process Python binding; and
- `ait-node` - direct Node-API binding and npm packaging component.

The seven components come from five independent source authorities. Each component retains its source Snapshot, artifact digest, and license identity. Matching version text alone does not replace release-family admission.

Public install channels

- GitHub Release provides the seven component families as canonical native, source, checksum, validation, and receipt assets.
- Homebrew installs the stable `ait-native` formula for supported macOS and Linux targets.
- PyPI publishes `ait-native==1.1.1` platform wheels containing `ait`, `ait-server`, and the direct Python binding.
- npm publishes `@wa120/ait-native@1.1.1` under the `latest` dist-tag and six exact-version Node-API platform packages.
- The signed apt repository publishes the `ait-native` bundle and a standalone `ait-runner` package in the `stable` suite for Debian and Ubuntu on `amd64` and `arm64`.
- GHCR publishes Linux `amd64` and `arm64` indexes for `ait-server` and `ait-runner`.

Package composition

Route	Admitted component content
GitHub Release	All seven components for the six native target triples admitted by the family manifest.
Homebrew `ait-native`	`ait` and `ait-server`; server installation does not start the service.
PyPI `ait-native==1.1.1`	`ait`, `ait-server`, and `ait-python`.
npm `@wa120/ait-native@1.1.1`	`ait-node` facade plus the matching platform addon; no server executable.
apt `ait-native`	`ait` and `ait-server`; server installation remains inactive.
apt `ait-runner`	Standalone `ait-runner`.

Route	Admitted component content
GHCR	Standalone `ait-server` and `ait-runner` images.

The optional `ait-agent` and `ait-agent-worker` executables are available from the GitHub Release. They are separate from the primary local coding-client experience and are not required for the normal `ait` workflow.

Publication state

The GitHub Release, PyPI, all seven npm packages, the Homebrew formula, the signed apt `stable` suite, and both attested GHCR images were published and independently read back for stable `1.1.1`. Each channel's default route (GitHub latest release, PyPI latest version, npm `latest` dist-tag, stable formula and suite) resolves to `1.1.1`, so unpinned installs receive the stable release.

License boundary

`ait`, `ait-agent`, `ait-agent-worker`, `ait-runner`, `ait-python`, and `ait-node` use Apache-2.0. `ait-server` uses AGPL-3.0-only. A bundle does not merge these licenses or erase component notices.

Exact release evidence

The page authority panel links the immutable `v1.1.1` source tag, GitHub Release, central distribution contract, and exact parser source. The version manifest also pins the five source Snapshots and the SHA-256 of both source documents. Use those authorities and the checksum-bound release assets when reproducing a build or validating an installed executable.

REFERENCE

Troubleshooting

Use read-only evidence first, preserve recoverable state, and follow exact AIT recovery hints.

AUDIENCE Developers and operators

https://ait-native.dev/technical/v1.1.1/troubleshooting/

Start with read-only evidence

These recovery steps apply to the version-pinned 1.1.1 command surface.

Do not guess whether the workspace, Task, or remote is healthy. Begin with the narrow read that matches the symptom.

```
ait status --json
ait diff --stat
ait queue summary
ait worktree status --json
```

Keep the exact error and next-action text. AIT decision surfaces intentionally report the command that can safely advance or inspect the current state.

The repository is not initialized

Run `ait init` from the intended repository root. If an `.ait` directory exists but is incomplete, do not replace it blindly; inspect the error and use the bounded repair option only when it matches the reported condition.

A regression needs attribution

Use `ait blame` on the smallest useful source range before selecting a repair.

```
ait blame <path> --line <number>
```

Record the repair in a new sprint item and Task. Completed lineage remains historical evidence.

A Task workspace is missing or stale

Inspect the Task and worktree before recreating materialization.

```
ait task audit <task-id>
ait worktree doctor --refresh --json
```

Follow the exact recovery command reported for that Task. Preserve unrelated workspace changes and avoid deleting a path that is not precisely identified as AIT-owned.

Remote readiness is pending

Run the text-only readiness surface again and follow its stated gate.

```
ait workflow ready <change-id> --apply
```

Local test success does not substitute for required remote CI. A failed review, policy, attestation, or CI gate must remain visible until its owning evidence is corrected.

The server or runner is unavailable

Check the configured remote, server health, repository index, runner compatibility, and repository-owned CI entrypoint. Do not switch a remote-backed Task to a different authority merely to bypass the failure.

Escalate without destroying evidence

When recovery requires a new user decision, stop with the current Task, Snapshot, and worktree intact. Report the exact identifier, failing gate, command output, and last successful validation.

APPENDIX

Appendix: Configuration Files

Map every public 1.1.1 AIT configuration file to its purpose, owner, mutation boundary, and detailed schema reference.

AUDIENCE Developers, operators, integrators, and release engineers

https://ait-native.dev/technical/v1.1.1/appendix/configuration-files/

Configuration-file map

1.1.1 uses a small set of repository files for durable user configuration. The files are independent contracts: a value in one file does not silently replace the authority owned by another file.

File	Purpose	Normal owner
`.ait/config.json`	Repository identity, Line and remote routing, workflow defaults, Task-worktree placement, and bounded runtime hints.	`ait init`, `ait config`, `ait remote`, and the owning AIT lifecycle.
`.ait/policy.yaml`	Admission requirements selected by the Repository policy profile.	`ait init`; review changes together with `.ait/config.json` `.policy_profile`.
`.aitignore`	Repository-relative visibility rules for workspace, status, Snapshot, and Plan filesystem discovery.	Repository author.
`ci/patch_ci.json`	Patchset CI suite catalog used for remote readiness.	Repository author; `ait remote add` can create a starter file without overwriting an existing one.
`ci/config.contract.json`	Optional local locator for a non-default CI suite catalog.	Repository author. This is not a replacement for the canonical catalog required at remote registration.
`ait-external.toml`	Direct external-Repository declarations and language bindings.	Repository author.

File	Purpose	Normal owner
`ait-external.lock`	Exact resolved external graph and Snapshot pins.	`ait external update`; commit the generated result.
`ait-external.links.toml`	Local development overrides for external materialization.	`ait external link` and `ait external unlink`; do not use for release-ready materialization.
`.ait/agent-workers.json`	Named Telegram, LINE, Discord, and Slack agent-worker instances.	`ait-agent <transport>` commands, or an operator following the schema.
`ait-release.json`	Generic command-adapter release package, components, commands, and artifacts.	Release author.
`ait-release-family.json`	Coordinated AIT native-family release matrix.	AIT release-family maintainer; this is a specialized AIT-family contract.
`.ait-worktree.json`	One managed worktree's overlay and binding metadata.	AIT only.

The server has no separate public `ait-server.json`, `ait-server.toml`, or `.ait/server.*` configuration file in 1.1.1. Its supported operator inputs are command-line options and the documented environment contract.

Which file should I edit?

- Prefer the command that owns a field. It can validate the value and preserve lifecycle invariants.
- Hand-edit only author-owned manifests such as `.aitignore`, `ci/patch_ci.json`, `ait-external.toml`, and the release manifests.
- Treat `ait-external.lock`, `ait-external.links.toml`, and `.ait-worktree.json` as generated or command-managed outputs.
- Keep credentials out of versioned files when a documented process or worker environment-file input is available.
- Verify JSON, YAML, or TOML syntax before starting a Task or remote registration. Unknown-field behavior differs by contract and is stated on each reference page.

What is intentionally outside this map?

Runtime state, caches, Binary DB authorities, activation receipts, generated artifacts, package-manager manifests, build-tool configuration, and agent instruction files are not user configuration for AIT itself. They have their own owners and must not be inferred from this appendix.

Related references

- Repository and worktree configuration
- Policy and ignore rules
- Patchset CI configuration
- External Repository configuration
- Agent-worker configuration
- Release manifests
- Environment variables

APPENDIX

Appendix: .ait/config.json

The complete current 1.1.1 persisted Repository configuration contract, including ownership, defaults, lifecycle writers, worktree overlays, and recovery boundaries.

AUDIENCE Developers, operators, and Repository owners

https://ait-native.dev/technical/v1.1.1/appendix/configuration-file/

Contract and authority

`.ait/config.json` is the authoritative persisted Repository configuration for 1.1.1. It stores Repository identity and routing, workflow defaults, remote registration, Task-worktree placement, and a small set of AIT-managed runtime hints. This page enumerates every current supported field and nested field.

Three JSON surfaces are related but are not interchangeable:

Surface	Authority and behavior
`.ait/config.json`	Persisted Repository-wide authority. AIT reads this file from the canonical Repository root.
`.ait-worktree.json`	AIT-managed overlay in one materialized worktree. Non-null overlay values replace the corresponding root values only for that worktree's effective runtime.
`ait config show --json`	Effective diagnostic projection after defaults, derivation, actor detection, remote resolution, and any worktree overlay. It is not a byte-for-byte rendering of either file.

The JSON reader may preserve an unrecognized key while rewriting another setting. Preservation does not make that key supported. Do not use arbitrary keys as extension points.

Safe mutation boundary

Use the command that owns a value whenever 1.1.1 exposes one:

```
ait config show --json
ait config set --workflow-mode solo_remote
ait config set --id-namespace-prefix W
ait config unset task-worktree-main-seed-ram-max-bytes
ait remote add origin https://server.example.invalid --default
ait doctor memory-root --json
```

`ait config set` admits ten user-owned settings and `ait config unset` admits eight optional overrides. Repository indexes, Line selectors, remote records, derived scopes, Plan binding internals, worktree overlays, and wait hints are not writable through `ait config set`. Let their owning AIT lifecycle write them.

If an advanced file-only value must be maintained, stop concurrent AIT mutations, preserve a trusted copy, write valid JSON atomically, and verify the result before resuming work. Never copy another Repository's `repository_index`, remote map, or worktree overlay.

Representative root file

This neutral sample shows the common shape; it is not a recovery template. The server-assigned index, remote metadata, paths, and identity must come from the Repository's own lifecycle.

```
{
  "repo_name": "sample-project",
  "default_line": "main",
  "current_line": "main",
  "default_remote": "origin",
  "repository_index": 42,
  "remotes": {
    "origin": {
      "remote_id": 1,
      "url": "https://server.example.invalid",
      "repo_name": "sample-project",
      "created_at": "2026-08-17T08:00:00Z"
    }
  },
  "id_namespace_prefix": "W",
  "policy_profile": "prototype",
  "default_author_mode": "ai_with_human_review",
  "workflow_mode": "solo_remote",
  "workflow_default_scope": "remote",
  "task_default_scope": "remote",
  "change_default_scope": "remote",
  "sprint": "on",
  "plan_task_binding": { "mode": "required" },
  "task_worktree": {
    "alias_root": ".ait-worktree-links",
    "main_seed_ram_max_bytes": 8589934592
  }
}
```

Fresh initialization

`ait init` creates `repo_name`, `default_line`, `current_line`, a null `default_remote`, an empty `id_namespace_prefix`, `policy_profile`, `default_author_mode`, `sprint`, and `plan_task_binding`. On a supported host it can also record a detected `task_worktree.memory_root`. Set an optional default model later with `ait config set --default-model <model>`.

The default Line is `main`. Without an explicit Repository name, the canonical root directory name is used. A fresh policy profile is one of `prototype`, `team`, or `release`; it must match `.ait/policy.yaml`.

Repository identity and Line selection

Field	JSON contract	Owner, absence, and mutation
`repo_name`	Nonempty string.	`ait init` writes it. If missing in an older Repository, runtime falls back to the canonical root directory name. It is not remote identity.
`default_line`	Nonempty Line-name string.	`ait init` writes it; absence falls back to `main`. Repository initialization owns changes.
`current_line`	Nonempty Line-name string.	AIT Line operations maintain the selected canonical-root Line. Absence falls back to `default_line`.
`default_remote`	Remote-name string or `null`.	`ait remote add ... --default` writes it. A non-null value must name an entry in `remotes`; absence means no default remote.
`repository_index`	Unsigned 32-bit integer.	The server allocates it and `ait remote add` persists it. Once present, 1.1.1 refuses to replace it with a newly returned index. There is no public setter.
`id_namespace_prefix`	String containing only ASCII letters or digits; stored uppercase. Empty is valid.	`ait init`, `ait config set --id-namespace-prefix`, or `ait config unset id-namespace-prefix`. The current server namespace is at most two characters; use the empty value or a value accepted by both client and server.

Policy, provenance, and local reviewer defaults

Field	JSON contract	Owner, absence, and mutation
`policy_profile`	`prototype`, `team`, or `release`.	Selected by `ait init`; it must equal the policy ID in `.ait/policy.yaml`. Do not change only one side.
`default_author_mode`	`human_only`, `human_with_ai_assist`, `ai_with_human_review`, or `ai_only_experimental`.	`ait init`, `ait config set`, or `ait config unset`. Absence resolves to `ai_with_human_review`.
`default_model`	Nonempty string when present.	`ait config set --default-model`; `ait config unset default-model` removes it. Absence is reported as null.
`user_name`	Nonempty string when present.	`ait config set` or `unset`. Sole configured identity for required or automatic functional Task review.
`user_email`	Nonempty string when present.	`ait config set` or `unset`. Used by non-Task-review actor identity surfaces.
`task_review`	Boolean: `true` means required; `false` means automatic.	`ait config set --task-review required` or `ait config set --task-review automatic` converts the word to this boolean. Absence resolves to automatic.

The effective actor can also come from `AIT_NATIVE_ACTOR`. Therefore the effective values shown by `ait config show --json` can differ from the stored identity fields.

Workflow preset coupling

Set the complete preset rather than editing its dependent fields separately:

workflow_mode	Stored scope values
solo_local	workflow_default_scope, task_default_scope, and change_default_scope are local.
solo_remote	workflow_default_scope, task_default_scope, and change_default_scope are remote.

ait config set --workflow-mode <value> writes workflow_mode and all three scope fields. It also writes sprint: "on" and plan_task_binding.mode: "required" unless --sprint off is supplied in the same command.

Field	JSON contract	Owner, absence, and mutation
workflow_mode	solo_local, solo_remote, or team_remote.	ait config set --workflow-mode. If absent or inconsistent, the effective projection is derived from the scopes and Plan binding and can report custom.
workflow_default_scope	local or remote.	Written by the workflow preset. Absence resolves to local.
task_default_scope	local or remote.	Written by the workflow preset. Absence inherits workflow_default_scope.
change_default_scope	local or remote.	Written by the workflow preset. Absence inherits workflow_default_scope.
sprint	on or off.	ait init or ait config set --sprint. When present, it is the effective authority for Plan binding.
plan_task_binding.mode	off, advisory, strict, or required.	Current public writers emit required for sprint on and off for sprint off. If sprint is absent, this object supplies the effective mode; if both are absent, 1.1.1 stages required.

Changing these defaults does not relocate or rewrite an existing Task, Change, Plan binding, or worktree. Existing work finishes under the contract recorded when it started.

Remote map

remotes is an object keyed by the local remote name. ait remote add is its public writer.

Field	JSON contract	Owner and behavior
remotes	Object whose keys are nonempty remote names.	Absent means no registered remotes. Each value must be an object.
remotes.<name>.remote_id	Signed integer.	AIT assigns the next local ordinal. Older entries without it are read using their deterministic map ordinal, but current writes include it.
remotes.<name>.url	Nonempty string.	Required server base URL supplied to ait remote add.
remotes.<name>.repo_name	Nonempty string or absent.	AIT records the canonical Repository directory name used for server registration.
remotes.<name>.created_at	RFC 3339 timestamp string.	AIT records registration time. Older records can omit it.

The remote URL can reveal private network topology. It is routing data, not a credential. Authentication tokens belong in the process secret boundary, not this file.

`task_worktree` object and `AIT_RAM`

Field	JSON contract	Owner, absence, and mutation
`task_worktree`	Object or `null`.	`ait init` can add detected memory-root data. Absence still allows host discovery and persistent-disk fallback.
`task_worktree.memory_root.kind`	`macos_ram_volume`, `linux_memory_root`, or `windows_ramdisk`.	Must match the host and the proof expected by `ait doctor memory-root --json`.
`task_worktree.memory_root.root`	Nonempty path string.	Memory-backed root. Operator-maintained; use an absolute path for unambiguous diagnostics and recovery.
`task_worktree.memory_root.volume_name`	Nonempty string or absent.	macOS-only volume label. Other kinds ignore it. When absent on macOS, AIT can derive it from the mount basename.
`task_worktree.memory_root.sector_count`	Positive integer or absent.	macOS RAM-disk capacity in 512-byte sectors. Absence uses 16,777,216 sectors, exactly 8 GiB. Other kinds ignore it.
`task_worktree.ephemeral_root`	Nonempty path string or absent.	File-only Task runtime root. Relative paths resolve from the canonical Repository root. Absence derives the runtime location from a valid memory root when available.
`task_worktree.alias_root`	Nonempty path string or absent.	`ait config set --task-worktree-alias-root`; relative paths resolve from the Repository root. Absence uses `.ait-worktree-links`.
`task_worktree.main_seed_ram_max_bytes`	Nonnegative integer or absent.	`ait config set` or `ait config unset task-worktree-main-seed-ram-max-bytes`. Absence means no configured Snapshot-size cutoff; it does not mean unlimited physical RAM.

Memory placement is evaluated when a new Task worktree is created. A Snapshot strictly larger than a configured main-seed limit uses Repository-internal persistent storage. Otherwise AIT uses a validated supported RAM root when it can and falls back to Repository-internal persistent storage when it cannot. The complete isolation, capacity, and recovery behavior is in Parallel Task Isolation.

AIT-managed operational fields

Field	JSON contract	Owner, absence, and behavior
`worktree_name`	Nonempty string or absent.	AIT temporarily binds the canonical root to an active managed worktree and clears the binding during cleanup. Do not hand-edit it.
`workflow_ready_poll_seconds`	Number.	AIT learns a rounded readiness wait estimate from recent remote history. Positive effective values are clamped to 5 through 900 seconds; `0` marks an attempted bootstrap with no usable sample. Absence means no cached estimate.
`workflow_land_poll_seconds`	Number.	Same AIT-owned estimate for remote Land completion. It has the same 5-through-900-second effective bounds and `0` bootstrap marker.

The wait values are hints for user feedback, not readiness, policy, or timeout authority. Removing or changing them does not make an operation complete.

Compatibility fields in existing files

These fields can remain in an upgraded Repository. They are documented so a backup or audit is intelligible; do not add them to a fresh 1.1.1 file.

Field	JSON contract	Current 1.1.1 meaning
`repo_id`	Older opaque string.	Preserved compatibility data. Current server routing identity is `repository_index`; `repo_id` must not be used to select remote authority.
`snapshot_binary_db_storage`	Existing scalar value.	The current runtime always uses Binary DB layout 1 and ignores the selector value. Omission has the same storage behavior.
`plan_binary_db_storage`	Existing scalar value.	Same fixed layout-1 behavior for Plan authority.
`remote_sync_binary_db_storage`	Existing scalar value.	Same fixed layout-1 behavior for remote-sync authority.
`plan_task_binding_mode`	Older scalar mode.	Used only by a bounded compatibility path. Current configuration writes and reads `plan_task_binding.mode`; do not add the scalar to a new file.

Complete worktree overlay

AIT creates `.ait-worktree.json` in a managed worktree and merges every non-null value over the root configuration for that worktree. A Task worktree uses the following fields:

Field	JSON contract	Meaning
`worktree_name`	Nonempty string.	Stable AIT-owned materialization name and cleanup identity.
`current_line`	Nonempty Line-name string.	Task feature Line selected inside this worktree.
`repo_root`	Absolute path string.	Canonical Repository root whose `.ait` authority is shared through the managed link.
`workspace_root`	Absolute path string.	Physical worktree root. It can differ from the user-facing alias path.
`created_at`	RFC 3339 timestamp string.	Materialization time.
`materialized_snapshot_id`	Snapshot ID string or absent.	Snapshot currently restored into the worktree; AIT updates it after restore and rebase operations.

The overlay is ownership metadata, not a portable user profile. Do not copy or edit it, and do not replace the managed `.ait` link with a separate authority.

Verification, backup, and recovery

After a supported change, verify both the effective projection and the relevant subsystem:

```
ait config show --json
ait remote list --json
ait status --json
ait doctor memory-root --json
```

Back up the entire `.ait` authority, not only `config.json`. The file contains the server-assigned Repository index and remote routing required to reconnect the local authority to its existing offsite copy. Redact server URLs, local paths, names, and email addresses before sharing diagnostics. The file should not contain passwords, bearer tokens, or integration secrets.

If local authority is lost, do not run `ait remote add` against an existing duplicate-named server Repository and assume it will reattach. Restore the trusted saved `repository_index`, `default_remote`, and `remotes` routing data as part of the saved local authority, verify it with `ait config show --json` and `ait repo show --remote origin --json`, then preview `ait remote recover-head --remote origin` before adding `--apply`. If the exact saved routing cannot be restored, stop instead of creating replacement authority.

Related

- Configuration-file map
- `ait config`
- Complete `ait config` command reference
- Parallel Task Isolation
- Remote Infrastructure
- Appendix: Environment Variables

APPENDIX

Appendix: Policy and Ignore Rules

Configure 1.1.1 admission requirements in .ait/policy.yaml and repository visibility in .aitignore with exact accepted fields and matching rules.

AUDIENCE Repository owners, developers, and coding agents

https://ait-native.dev/technical/v1.1.1/appendix/repository-policy-ignore/

Repository policy and ignore rules

This page documents the two repository-root text files that control admission requirements and filesystem visibility. They solve different problems: `.ait/policy.yaml` decides which evidence is required; `.aitignore` decides which workspace paths AIT sees.

`.ait/policy.yaml`

`ait init` creates this file with owner-only permissions. Its `policy_id` must match `.ait/config.json.policy_profile`. 1.1.1 supports the policy IDs `prototype`, `team`, and `release`.

```
version: 1
policy_id: prototype
defaults:
  require_attestation: true
  require_tests: true
  require_lint: false
  require_security_scan: false
  require_license_scan: false
  require_ai_provenance: false
  require_code_review_summary: false
class_overrides:
  - when:
      content_class: docs_only
    set:
      require_tests: false
      require_lint: false
      require_security_scan: false
      require_license_scan: false
```

Root and requirement fields

Field	Type and meaning
`version`	Integer `1`.
`policy_id`	One of `prototype`, `team`, or `release`; must equal the Repository policy profile.
`defaults`	Required map containing the baseline requirement switches.
`class_overrides`	Optional ordered list of conditional partial overrides. An empty list is valid.

Every requirement value is an exact YAML boolean:

Requirement	Evidence controlled
`require_attestation`	An admitted attestation is required.
`require_tests`	Test evidence must pass.
`require_lint`	Lint evidence must pass.
`require_security_scan`	Security-scan evidence must pass.
`require_license_scan`	License-scan evidence must pass.
`require_ai_provenance`	AI provenance evidence is required.

Requirement	Evidence controlled
`require_code_review_summary`	A code-review summary is required.

The same seven names are accepted under an override's `set` map. `defaults` must contain all seven; `set` is partial and changes only the fields it names.

Override selectors

Each `class_overrides[]` entry requires a nonempty `when` map and a nonempty `set` map. `when` accepts either or both selectors:

Selector	Accepted values
`content_class`	`docs_only`, `code_change`
`author_class`	`human_only`, `ai_related`

Overrides are evaluated in file order. Keep conditions explicit and avoid overlapping entries whose result depends on ordering.

Generated profile baselines

All three generated profiles require attestation and tests and leave AI provenance and code-review-summary requirements off. Their other defaults are:

Profile	Lint	Security scan	License scan
`prototype`	false	false	false
`team`	true	false	false
`release`	true	true	true

The generated documentation-only override turns tests, lint, security scan, and license scan off. Changing a generated policy is an admission-policy change; review it together with the matching profile setting.

YAML acceptance rules

- Use spaces, never tabs, with two-space indentation steps.
- Use exact lowercase `true` and `false` values.
- Duplicate and unknown fields fail validation.
- Keep `version`, `policy_id`, `defaults`, and any override map at the exact nesting shown above.
- A first remote registration in 1.1.1 admits the exact `prototype` baseline and its complete documentation-only override. Select that profile before registering a new Repository. Locally configured `team` and `release` profiles remain valid policy files for their supported local use.

`.aitignore`

`.aitignore` is a line-oriented repository visibility contract. It applies to workspace inspection and to the files considered by status, Snapshot, and Plan filesystem discovery. It augments AIT's built-in exclusions; it does not replace them.

```
# Generated outputs
dist/
*.tmp

# Keep one fixture visible
!fixtures/expected.tmp

# Literal leading markers
\#notes.txt
\!important.txt
```

Rule syntax

Form	Behavior
blank line	Ignored.
`# comment`	Ignored.
`\#name`	Matches a literal leading #.
`!pattern`	Negates a prior match and makes the path visible again.
`\!name`	Matches a literal leading `!`.
`./path`	The leading `./` is removed.

Form	Behavior
`/path`	Anchored to the Repository root.
`path/`	Directory-only rule.
`name`	With no slash, matches that basename at any depth.
`*`	Matches any sequence within a path segment.
`?`	Matches one character within a path segment.

Rules are evaluated from top to bottom and the last matching rule wins. Other regular-expression characters are treated literally; character-class syntax is not part of the 1.1.1 matcher. `.aitignore` itself remains visible so its effect can be captured and reviewed.

An ignore rule changes visibility, not ownership or deletion. It does not erase an existing Snapshot, authorize secret storage, or make a generated directory safe to commit.

Verification

After changing either file, use read-only inspection before normal work:

```
ait config show --json
ait status
ait diff
```

For remote registration, run the actual `ait remote add` command only after the policy and Patchset CI catalog are ready; its validation is authoritative.

Related references

- Configuration-file map
- Repository and worktree configuration
- Patchset CI configuration

APPENDIX

Appendix: Patchset CI Configuration

Define the canonical 1.1.1 Patchset CI suite catalog, supported runners, checks, artifacts, and bounded local locator.

AUDIENCE Repository owners, CI maintainers, and operators

https://ait-native.dev/technical/v1.1.1/appendix/patchset-ci/

Patchset CI configuration

`ci/patch_ci.json` is the canonical 1.1.1 suite catalog used when a Repository is registered with an AIT server and when remote readiness requests Patchset CI. The catalog must contain at least one blocking Patchset gate.

Minimal command-bundle catalog

```
{
  "schema_version": 1,
  "suites": [
    {
      "schema_version": 1,
      "suite_id": "patchset_gate",
      "display_name": "Patchset Gate",
      "plane": "patchset",
      "mode": "gate",
      "default_blocking": true,
      "purpose": "Validate this repository before remote land.",
      "runner": {
        "kind": "command_bundle",
        "commands": ["./ait.sh test"]
      },
      "artifacts": {
        "log_path": ".ait/generated/ci/patchset_gate.log",
        "summary_json": ".ait/generated/ci/patchset_gate.json"
      }
    }
  ]
}
```

`ait remote add` creates a starter catalog when the file is missing and then stops so the placeholder command can be replaced. It never overwrites an existing catalog. The placeholder `replace-with-your-ci-command` is rejected.

Catalog and suite fields

Field	Contract
`schema_version`	Top-level integer `1`; required. The starter also records `1` on each suite.
`suites`	Array of suite objects. At least one object must be a named, runnable blocking Patchset gate.
`suites[].suite_id`	Nonempty unique operational name. Required for every selected blocking gate.
`suites[].display_name`	Optional human-readable string.
`suites[].plane`	Use `patchset` for Patchset CI selection.
`suites[].mode`	Use `gate` for readiness selection.

Field	Contract
`suites[].default_blocking`	Boolean. `true` makes the selected suite part of readiness evidence; `false` is informational.
`suites[].purpose`	Optional explanatory string.
`suites[].runner`	Required object with a nonempty supported `kind`.
`suites[].artifacts.log_path`	Optional relative log-artifact path.
`suites[].artifacts.summary_json`	Optional relative JSON-summary path.

Only suites whose `plane` and `mode` resolve to `patchset` and `gate` are planned. A catalog used for first registration needs at least one such suite with `default_blocking: true`. Do not rely on unknown keys as extension points.

`command_bundle` runner

This runner executes shell command strings in order and stops after the first failure. Prewarm and main commands share the admitted environment.

Runner field	Type and behavior
`kind`	Exact string `command_bundle`.
`commands`	Nonempty array of nonempty shell-command strings.
`prewarm_commands`	Optional array executed before `commands`.
`env`	Optional object of string environment values added to the clean CI process environment.
`runner_parallelism`	Optional positive integer used for admitted command parallelism.
`cpu_tokens`	Alias for `runner_parallelism`.

Runner field	Type and behavior
`workers`	Lower-precedence alias for `runner_parallelism`.
`timeout_seconds`	Optional positive bounded timeout applied to each process.

The scheduler can supply a stricter admitted parallelism value. Configuration does not grant CPU capacity by itself. Full logs and the bounded JSON summary are separate artifacts.

`test_discovery_sharded` runner

This runner discovers test cases, builds the selected test executables, and runs bounded shards. It supports Cargo and a generic command adapter.

Common fields

Runner field	Type and behavior
`kind`	Exact string `test_discovery_sharded`.
`adapter`	`cargo` by default; `command` selects the generic adapter.
`env`	Optional string-to-string environment object.
`runner_parallelism`, `cpu_tokens`, `workers`	Positive-integer parallelism names, in that precedence order.
`timeout_seconds`	Optional positive bounded process timeout.
`exclude_test_cases`	Optional array of exact discovered test-case names to omit.

Runner field	Type and behavior
`checks`	Optional pre-test check array described below.

Cargo-adapter fields

Runner field	Type and behavior
`cargo_binary`	Cargo executable name or path; default `cargo`.
`manifest_path`	Normalized relative path; default `Cargo.toml`.
`workspace`	Boolean; default `true`.
`build_args`	Optional argv strings replacing the default Cargo build arguments.
`doc_tests`	Boolean; default `false`. Not accepted with the command adapter.
`doc_test_args`	Optional argv strings for documentation tests.

Command-adapter fields

Runner field	Type and behavior
`discovery_program`	Required nonempty executable name or path.
`discovery_args`	Optional discovery argv array.
`discovery_output_format`	`json_array` by default, or `lines`.
`run_program`	Required nonempty executable name or path.
`run_args`	Optional run argv array.
`append_test_items`	Boolean; when true, discovered item names are appended to the run argv. Default `false`.

Runner field	Type and behavior
`working_directory`	Normalized relative directory inside the workspace; default is the workspace root. It must exist.

Pre-test checks

`checks` entries accept `check_id`, `kind`, `file_name_suffix`, `exclude_dirs`, and `args`. Missing `check_id` becomes `check-N`.

- `kind: "forbid_files"` fails if a file ends with `file_name_suffix` outside the named directory components. The suffix defaults to `.py`.
- `kind: "cargo_fmt"` runs Cargo formatting verification. `args` can replace its default argv.

Other check kinds are rejected when the suite runs.

Optional build cache

`runner.build_cache` accepts `policy` and `executable_manifest_path`. The only enabling policy is `reuse_when_rust_inputs_unchanged`; absence or another value keeps reuse disabled. Changed-path evidence is supplied by the execution request and should not be authored as static catalog data.

Optional local locator: `ci/config.contract.json`

When `ci/patch_ci.json` is absent, local workflow command discovery can read:

```
{
  "schema_version": 1,
  "ci": {
    "suite_manifest_path": "config/patchset-suites.json"
  }
}
```

1.1.1 reads only the nonempty `ci.suite_manifest_path` value and uses it only if the referenced file exists. A present `ci/patch_ci.json` always wins. Remote Repository registration still validates and publishes the canonical `ci/patch_ci.json`; the locator does not redirect that registration contract.

Operational checks

```
ait remote add origin https://server.example.invalid --default
ait workflow ready <change-id> --apply
ait task audit <task-id>
```

`ait workflow ready` is the decision surface for the current Patchset. Passing a locally invoked test command does not substitute for the server evidence required by a remote Task.

Related references

- Configuration-file map
- Repository policy and ignore rules
- Remote infrastructure

APPENDIX

Appendix: External Repository Configuration

Declare, lock, materialize, bind, and locally override external Repositories through the three 1.1.1 TOML contracts.

AUDIENCE Developers, integrators, and release engineers

https://ait-native.dev/technical/v1.1.1/appendix/external-repositories/

External Repository configuration

1.1.1 separates author intent, exact resolution, and local development overrides into three TOML files:

File	Authority
`ait-external.toml`	Direct declarations authored by the Repository.
`ait-external.lock`	Complete resolved graph with exact Snapshot pins; generated by AIT.
`ait-external.links.toml`	Local path overrides managed by AIT commands.

`ait-external.toml`

Each `[[external]]` row identifies one direct external Repository and where its content is materialized.

```
[[external]]
name = "shared-core"
repo_name = "shared-core"
repository_index = 23
remote = "origin"
line = "main"
snapshot = "SNP-0123456789AB"
materialize_to = ".ait-external/shared-core"
license = "Apache-2.0"
version = "1.2.0"

[external.bindings.rust]
kind = "cargo-path"
path = "rust/crates/shared-core"
package = "shared-core"

[external.bindings.python]
kind = "python-path"
path = "python"
package = "shared-core"
module = "shared_core"
```

Declaration fields

Field	Type and meaning
`name`	Required nonempty local external name. It identifies the declaration and link override.
`repo_name`	Required nonempty source Repository name.
`repository_index`	Required unsigned 32-bit server Repository index.
`remote`	Required nonempty configured remote name used to resolve the source.
`line`	Required nonempty source Line.
`snapshot`	Required nonempty exact source Snapshot ID. This is the pin, not merely a branch hint.

Field	Type and meaning
`materialize_to`	Required normalized Repository-relative destination. Absolute paths and `..` traversal are rejected.
`license`	Required nonempty declared license expression or label.
`version`	Optional nonempty human-facing version string. The Snapshot remains the content authority.
`bindings`	Optional map with at most one binding for each supported language.

Direct names must be unambiguous. Materialization and binding paths are validated as repository-relative paths and cannot escape their owning root.

Binding fields

Table	Required fields	Optional metadata
`[external.bindings.rust]`	`kind = "cargo-path"`, `path`	`package`
`[external.bindings.python]`	`kind = "python-path"`, `path`	`package`, `module`
`[external.bindings.node]`	`kind = "file-package"`, `path`	`package`
`[external.bindings.go]`	`kind = "replace-path"`, `path`	`module`

Each path is relative to the external Repository materialization. Metadata, when present, must be nonempty. Binding validation can also check the relevant language tool and dependency file; a valid TOML shape alone does not prove a usable package binding.

`ait-external.lock`

Do not synthesize or casually hand-edit this file. `ait external update` resolves direct and transitive externals, normalizes ordering, validates the graph, and writes the lock atomically.

```
format = "ait.external.lock"

[[node]]
name = "shared-core"
repo_name = "shared-core"
repository_index = 23
remote = "origin"
line = "main"
snapshot = "SNP-0123456789AB"
parent_path = ""
materialize_to = ".ait-external/shared-core"
license = "Apache-2.0"
version = "1.2.0"

[[node.binding]]
language = "rust"
kind = "cargo-path"
path = "rust/crates/shared-core"
package = "shared-core"
```

Lock fields

Field	Contract
`format`	Exact string `ait.external.lock`.
`[[node]]`	Zero or more normalized direct and transitive graph nodes.
`node.name`	Required nonempty name, unique together with `parent_path`.
`node.repo_name`	Required nonempty source Repository name.
`node.repository_index`	Required unsigned 32-bit source Repository index.
`node.remote`	Required nonempty remote name.

Field	Contract
`node.line`	Required nonempty source Line.
`node.snapshot`	Required nonempty exact Snapshot ID.
`node.parent_path`	Empty for a direct root node; otherwise a normalized relative ancestry path.
`node.materialize_to`	Required normalized relative destination.
`node.license`	Required nonempty license label.
`node.version`	Optional version string.

Field	Contract
`[[node.binding]]`	Zero or more normalized binding summaries.
`node.binding.language`	`rust`, `python`, `node`, or `go`.
`node.binding.kind`	The exact kind admitted for that language.
`node.binding.path`	Normalized relative path.
`node.binding.package`	Optional nonempty package name.
`node.binding.module`	Optional nonempty module name.

The lock detects missing, extra, and field-drifted direct roots against `ait-external.toml`. A clean lock must be committed with the manifest so every worktree and remote build sees the same graph.

`ait-external.links.toml`

Local links let a developer temporarily materialize one named external from an existing directory instead of its locked Snapshot.

```
[[link]]
name = "shared-core"
path = "../shared-core"
```

Each row has exactly the operational values `name` and `path`. The name refers to an external declaration; the path must resolve to an existing directory. Use the owning commands:

```
ait external link shared-core ../shared-core
ait external unlink shared-core
```

AIT removes the file when the final link is removed. Local links are a developer override and are rejected by locked or release-ready materialization. They never modify the declared Snapshot or generated lock.

Update and verification flow

```
ait external update
ait external status
ait external doctor
ait diff ait-external.toml ait-external.lock
```

Review a lock change like a dependency update: verify Repository index, Line, Snapshot, materialization destination, license, and binding drift before recording the resulting Snapshot.

Related references

- Configuration-file map
- Complete CLI reference
- Parallel Task isolation

APPENDIX

Appendix: Agent-worker Configuration

Configure named Telegram, LINE, Discord, and Slack workers, credential precedence, runtime paths, transport modes, and local reply behavior.

AUDIENCE Agent operators and integration maintainers

https://ait-native.dev/technical/v1.1.1/appendix/agent-workers/

Agent-worker configuration

`.ait/agent-workers.json` declares named messaging workers for one AIT Repository. 1.1.1 supports `telegram`, `line`, `discord`, and `slack`. The default location can be overridden with `AIT_AGENT_CONFIG_PATH`.

Read ait-agent-worker: Messaging and Reply Runtime first for the executable's role, complete command surface, message-to-reply path, transport modes, native Codex provider, runtime admission, and security limits. This appendix is the field-by-field authoring reference.

Canonical document shape

```
{
  "version": 1,
  "workers": {
    "telegram/support": {
      "kind": "telegram",
      "name": "support",
      "mode": "poll",
      "username": "support_bot",
      "env_path": ".ait/agent-runtime/telegram.env",
      "request_timeout_seconds": 30,
      "local_reply": {
        "model": "gpt-5.4-mini",
        "reasoning_effort": "low",
        "sandbox": "workspace-write",
        "turn_timeout_seconds": 300
      }
    }
  }
}
```

The author file is the direct `version` plus `workers` object shown above. An internal diagnostic projection may wrap it as `config`; do not write that wrapper into the repository file.

Worker keys use exact `<kind>/<name>` form. The kind and name must be nonempty, the name must not contain `/`, and an explicit `kind` must match the key. `version` is integer `1` for the current contract. Invalid normalization issues cause worker startup to fail closed.

Common worker fields

Field	Type and behavior
`kind`	`telegram`, `line`, `discord`, or `slack`; normally derived from the key.
`name`	Nonempty instance name; normally derived from the key.
`runtime_root`	Optional path for worker runtime files; default `.ait/agent-runtime`. Relative paths resolve from the Repository root; ~ uses the process home directory when available.
`sync_state_path`	Optional state-file path. Default `<runtime_root>/<kind>-<safe-name>-sync.json`.
`env_path`	Optional credential environment-file path. Default `<runtime_root>/<kind>.env`.
`web_url`	Optional normalized browser base URL associated with replies.

Field	Type and behavior
`request_timeout_seconds`	Optional positive request timeout. `inf`, `infinite`, or `none` disables supported generic timeouts; Telegram also accepts `null` and `unlimited`.
`local_reply`	Optional local reply-provider object, documented below.
`created_at`, `updated_at`	Command-managed timestamps.
`pid_file`, `log_file`, `termination_context_path`	Command-managed lifecycle paths when present; do not use them to configure reply behavior.

The runtime target is derived from `.ait/config.json`: `solo_local` runs locally; a remote workflow requires a valid default remote URL. The Repository `default_model` supplies the Telegram model fallback when no worker model is set.

Credential sources and precedence

Credential fields must be nonempty strings. Resolution order is:

1. the worker entry;
2. the current process environment;
3. the worker `env_path` file.

Other behavior fields come from the worker entry or their compiled default; the env file does not override them. Environment files accept KEY=VALUE lines, blank lines, # comments, and one matching pair of single or double quotes around a value. They are not shell scripts.

Keep secrets out of a committed manifest. Prefer an owner-readable env file or the process environment, and use the exact variable names in the environment appendix.

`local_reply`

Unknown fields in this nested object are rejected.

Field	Contract
`program`	Optional nonempty reply-provider executable.
`args`	Optional array of strings without NUL characters.
`timeout_seconds`	Optional finite positive number.
`append_turn_analysis`	Optional boolean.
`codex_program`	Optional nonempty Codex executable override.
`model`	Optional nonempty model name.

Field	Contract
`reasoning_effort`	`none`, `minimal`, `low`, `medium`, `high`, `xhigh`, `max`, or `ultra`.
`sandbox`	`read-only`, `workspace-write`, or `danger-full-access`.
`turn_timeout_seconds`	Positive number, numeric string, or `inf`, `infinite`, `none`, or `unlimited`.

Sandbox choice is an execution boundary, not a messaging preference. Grant only the access required by the worker's Repository task.

Telegram fields

Field	Contract and default
`token`	Required bot token, or supply `AIT_TELEGRAM_BOT_TOKEN`/`BOT_TOKEN`.
`username`	Optional; a leading `@` is removed.
`mode`	`poll` (default) or `webhook`.
`bind_host` / `bind_port`	Webhook listener; defaults `127.0.0.1` and `8090`.
`webhook_path`	HTTP path; default `/webhook`.
`webhook_secret`	Optional secret or `AIT_TELEGRAM_WEBHOOK_SECRET`.

Field	Contract and default
`poll_timeout_seconds`	Integer at least 5; default 45.
`background_sync_enabled`	Boolean; default false.
`background_sync_interval_seconds`	Number at least 5; default 30.
`openai_api_key`	Optional worker credential; placeholder values are rejected.
`openai_base_url`	Optional normalized base URL; default `https://api.openai.com/v1`.
`openai_model`	Optional model; falls back to the Repository default model, then `gpt-5.4-mini`.

Field	Contract and default
`openai_reasoning_effort`	Optional string; default `low`.
`openai_timeout_seconds`	Optional positive timeout; inherits the request timeout when omitted.
`openai_max_output_tokens`	Integer at least 64; default 700.
`turn_merge_window_seconds`	Nonnegative number; default 0.35.
`turn_merge_max_messages`	Positive integer; default 4.
`decoupled_reply_enabled`	Boolean; default true.

Field	Contract and default
`reply_markdown_enabled`	Boolean; default true.
`owner_bootstrap_enabled`	Boolean; default true.
`stt_mode`	`off` (default) or `local-stt`.
`stt_model`	Local speech-to-text model name.
`stt_device`	Device selector; default `auto`.
`stt_compute_type`, `stt_language`	Optional strings.

Field	Contract and default
`stt_include_audio_uploads`	Boolean; default false.
`stt_program`	Optional local speech-to-text executable path.
`stt_timeout_seconds`	Positive number from 0.1 through 3600; default 120.
`expected_concurrent_workers`, `workers_per_shard`	Optional positive integers.
`event_loop_backend`	Optional backend string.

LINE fields

Field	Contract and default
`token`	Required channel access token, or `AIT_LINE_CHANNEL_ACCESS_TOKEN`/`LINE_CHANNEL_ACCESS_TOKEN`.
`secret`	Required channel secret, or `AIT_LINE_CHANNEL_SECRET`/`LINE_CHANNEL_SECRET`.
`api_base_url`	Default `https://api.line.me`.
`bind_host` / `bind_port`	Defaults `127.0.0.1` and `8091`.
`webhook_path`	Default `/callback`.

Discord fields

Field	Contract and default
`application_id`	Required string, or `AIT_DISCORD_APPLICATION_ID`/`DISCORD_APPLICATION_ID`.
`public_key`	Optional credential for interaction verification.
`bot_token`	Optional credential for gateway or API use.
`turn_timeout_seconds`	Optional positive timeout; default is at least 300 seconds when request timeout is set.
`api_base_url`	Default `https://discord.com/api/v10`.
`http_user_agent`	Default `curl/8.7.1`.

Field	Contract and default
`bind_host` / `bind_port`	Defaults `127.0.0.1` and `8092`.
`interaction_path`	Default `/interactions`.

The selected worker command determines whether `public_key`, `bot_token`, or both are required for that service mode.

Slack fields

Field	Contract and default
`app_token`	Optional credential for socket mode.
`signing_secret`	Optional credential for HTTP command verification.
`api_base_url`	Default `https://slack.com/api`.
`http_user_agent`	Default `ait-agent-worker/0.1`.
`bind_host` / `bind_port`	Defaults `127.0.0.1` and `8093`.
`command_path`	Default `/command`.

Field	Contract and default
`ack_text`	Default `ait is thinking...`.
`response_type`	Default `in_channel`.

The selected Slack command determines whether socket credentials or an HTTP signing secret is mandatory.

Verification

Use the agent-worker inspection and launch commands from the complete CLI reference. Diagnostic output reports only whether credentials are set; it does not print their values. Test webhook listeners on loopback before exposing them through a trusted reverse proxy.

APPENDIX

Appendix: Release Manifests

Expand the generic command-adapter and specialized AIT native-family release manifest schemas, bounds, and ownership rules.

AUDIENCE Release engineers and package maintainers

https://ait-native.dev/technical/v1.1.1/appendix/release-manifests/

Release manifests

1.1.1 has two release manifest contracts with different scope:

File	Profile and purpose
`ait-release.json`	`generic-command`: reusable release adapter for one package with one or more components.
`ait-release-family.json`	`family`: specialized coordinator for the official AIT native release family.

Both files are read from the recorded release-source Snapshot. Unknown fields are rejected, paths are normalized repository-relative paths, and declarations are checked against the files actually present in that Snapshot.

Generic adapter: `ait-release.json`

```
{
  "schema": "ait.release.adapter/v1",
  "package": {
    "name": "sample-tool",
    "version": "1.2.0",
    "description": "Portable sample package.",
    "license_files": [
      {"path": "LICENSE", "role": "license"},
      {"path": "NOTICE", "role": "notice"}
    ]
  },
  "components": [
    {
      "id": "cli",
      "ecosystem": "native",
      "working_directory": ".",
      "dependency_files": ["Cargo.toml", "Cargo.lock"],
      "commands": {
        "prepare": [],
        "test": [["cargo", "test", "--locked"]],
        "build": [["cargo", "build", "--release", "--locked"]],
        "smoke": [["target/release/sample-tool", "--version"]]
      },
      "artifacts": [
        {"path": "target/release/sample-tool", "kind": "native-executable"}
      ]
    }
  ]
}
```

Root and package fields

Field	Contract
`schema`	Exact string `ait.release.adapter/v1`.
`package`	Required object.
`package.name`	Required nonempty bounded single-line string.
`package.version`	Required nonempty bounded single-line string.
`package.description`	Optional string or `null`.
`package.license_files`	Optional array of one through eight rows.

Field	Contract
`license_files[].path`	Normalized relative path to a file in the release source. Paths must be unique.
`license_files[].role`	Exact `license` or `notice`; each role can occur at most once.
`components`	Required array of one through 64 component objects.

Component fields

Field	Contract
`id`	Required identifier, unique across components.
`ecosystem`	Required identifier describing the component toolchain.
`working_directory`	Required normalized relative directory, or `.` for the root. It must exist in the source Snapshot.
`dependency_files`	One through 64 unique normalized paths relative to `working_directory`; every file must exist.
`commands`	Required object containing only `prepare`, `test`, `build`, and `smoke`.
`artifacts`	One through 128 artifact rows.

Command phases

Each command is an argv array, not a shell string. `test` and `build` are required and each must contain one through 16 commands. `prepare` and `smoke` are optional and may contain zero through 16 commands. Every command contains one through 64 string arguments; the executable argument cannot be blank.

Commands run directly without an implicit shell. 1.1.1 replaces only an exact whole argv value matching one of these tokens:

```
$AIT_RELEASE_ID
$AIT_RELEASE_VERSION
$AIT_RELEASE_COMPONENT
$AIT_RELEASE_ECOSYSTEM
$AIT_RELEASE_TARGET
$SOURCE_DATE_EPOCH
```

The tokens are not substring interpolation and do not authorize arbitrary environment expansion. Newlines, carriage returns, and NUL characters are rejected in argv values.

Artifact fields

Field	Contract
`path`	Required normalized relative output path, unique within the component.
`kind`	Required identifier such as `native-executable`, `python-wheel`, or another adapter-defined kind.
`target`	Optional identifier. Absence marks a portable artifact; a value associates the artifact with one release target.

The selected build must produce every declared artifact as an exact regular file. A target selection includes matching target artifacts; the portable selection includes only artifacts without a target.

Generic bounds

The manifest is limited to 1 MiB. Identifiers start with an ASCII letter or digit and then use letters, digits, `.`, `_`, or `-`. Text fields are bounded single-line strings. Relative paths use `/`, cannot be absolute, and cannot contain empty, `.`, or `..` segments, drive prefixes, colons, or backslashes.

AIT family coordinator: `ait-release-family.json`

This is not a generic multi-project manifest. Contract `ait.release.family/v3` coordinates the official AIT native components, targets, public-source projection, and distribution identities.

```
{
  "schema": "ait.release.family/v3",
  "family": {
    "name": "ait-native",
    "version": "1.1.1",
    "channel": "stable",
    "tag": "v1.1.1"
  },
  "targets": ["aarch64-apple-darwin", "x86_64-unknown-linux-gnu"],
  "public_source": {},
  "components": [],
  "distributions": [],
  "compatibility": {}
}
```

The empty objects and arrays above show the root shape only; they are not a valid release. A real family manifest must satisfy all relationships below.

Family identity and target matrix

Field	Contract
`schema`	Exact string `ait.release.family/v3`.
`family.name`	Required identifier.
`family.version`	Stable `MAJOR.MINOR.PATCH` or RC `MAJOR.MINOR.PATCH-rc.N`, matching `channel`.
`family.channel`	Exact `rc` or `stable`.
`family.tag`	Exact `v` plus `family.version`.
`targets`	One through 32 unique target identifiers.

Components

`components` contains one through 64 unique component objects.

Field	Contract
`id`	Required unique component identifier.
`source_repository`	Required source Repository identifier. In v3 public-source mode it must be one of `ait-core`, `ait-server`, `ait-runner`, `ait-python`, or `ait-node`.
`source_snapshot`	Required valid exact Snapshot ID.
`ecosystem`	Required identifier.
`license`	Required bounded SPDX-style expression.
`version_scheme`	`family` or `pep440`.

Field	Contract
`version`	Must equal the family version for `family`, or its canonical PEP 440 mapping for `pep440`.
`artifacts`	One through 32 unique artifact requirements.
`artifacts[].kind`	Required artifact-kind identifier.
`artifacts[].targets`	Unique target identifiers from the root matrix. An empty array means one portable artifact of that kind.

The tuple of artifact kind and target must not be duplicated within a component.

Distributions

`distributions` contains one through 128 rows. Every component must be covered by at least one distribution.

Field	Contract
`channel`	`github`, `pypi`, `npm`, `oci`, `homebrew`, `apt`, or `winget`.
`role`	`product`, `standalone`, or `implementation`.
`identity`	Required bounded distribution identity; the channel/identity pair must be unique.
`components`	One through 64 declared component IDs.
`targets`	One through 32 targets from the family matrix. Each selected component must provide that target or a portable artifact.

`public_source`

The v3 public-source object is deliberately exact because it produces one reviewable release monorepo.

Field	Contract
`model`	Exact `release-monorepo`.
`identity`	Exact official public-source identity `weita2026/ait-native`.
`product_document`	Exact `docs/distribution.md`.
`family_manifest`	Exact `ait-release-family.json`.
`mapping_manifest`	Exact `ait-monorepo-source.json`.
`build_entrypoints.unix`	Exact `build-release.sh`.

Field	Contract
`build_entrypoints.windows`	Exact `build-release.ps1`.
`build_entrypoints.implementation`	Exact `build-release.mjs`.
`subtrees`	One row for every source Repository used by the components.
`subtrees[].source_repository`	Declared source Repository.
`subtrees[].path`	Exact same-named public directory.
`subtrees[].transforms`	Ordered allowlisted transform IDs applied to that subtree.

Field	Contract
`transforms`	Exact transform definitions required by the selected source Repository set.
`transforms[].id`	Bounded ASCII ID ending in `/v1`.
`transforms[].source_repository`	Owning source Repository.
`transforms[].path`, `from`, `to`	Exact allowlisted path rewrite tuple for that transform ID.

The current allowlist contains only the sibling-core path projections required by the runner and Python source subtrees. Arbitrary source rewriting is not a manifest extension mechanism. The one GitHub product distribution must use the same public identity and cover every family component and target.

Compatibility map

`compatibility` is a required object with at most 64 entries. Each key is an identifier and each value is a nonempty bounded single-line string. It records consumer-facing compatibility facts; it does not replace component versions, Snapshot pins, or the target matrix.

Review boundary

- Record manifest edits in the same Snapshot as the referenced source and dependency-file changes.
- Never update a component `source_snapshot` without reviewing its admitted artifacts, license, version, and distribution coverage.
- Keep generated candidate, check, build, frozen, promotion, checksum, mapping, and receipt files out of this authoring reference. They are release outputs, not additional user configuration files.
- Use the complete `ait release` command reference to create, check, build, and inspect release evidence; do not hand-author output receipts.

Related references

- Configuration-file map
- Complete CLI reference
- Distribution and release

APPENDIX

Appendix: Environment Variables

The normalized 31-entry installed 1.1.1 environment contract, with explicit ownership, setting, secret, and process-boundary behavior.

AUDIENCE Developers, operators, integrators, and release engineers

https://ait-native.dev/technical/v1.1.1/appendix/environment-variables/

What this appendix covers

This is the complete installed process-environment contract for the current `ait-native 1.1.1` family: `ait`, Agent workers, `ait-server`, and `ait-runner`. It contains **31 normalized names or bounded families**. The entries come from the three machine-readable runtime registries, not from a search for every `AIT_` token in source files.

The contract stays in one document. Each row says which component owns the value, who sets it, whether it is secret, and what it changes.

- **User** and **operator** values may be placed in the launched process environment. Explicit command options still win where a command exposes the same choice.
- **AIT-injected** values cross a managed child-process boundary. A user should not set them to redirect another Task or CI attempt.
- **Automation** values belong only to the named supported build or release boundary.
- A secret belongs in a service or CI secret store. Do not commit it, print it, or place it in retained command-line history.

Neither `ait-server` nor `ait-runner` automatically loads a project `.env` file. A shell, service manager, container runtime, or the Agent supervisor must place configured values into the process environment.

Registry accounting

Registry	Published accounting
`ait.environment-contract/v1`	23 `ait-core`, CLI, Agent, and release-boundary entries.
`ait.server.environment-contract/v1`	7 server entries. `AIT_NATIVE_SERVER_DATA` and `AIT_PERFETTO_TRACE` are already owned by core, so the server contributes 5 new names.
`ait.runner.environment-contract/v1`	4 runner entries. Its external-repository family includes core's concrete `AIT_EXTERNAL_CORE_REPO_ROOT` input, so the runner contributes 3 additional normalized names.
Published total	**31 unique names or bounded families.** The runner's bounded external-Repository family contains the concrete core diagnostic member, so it is counted once.

Runtime, repository, and identity

Name	Owner, setting, secret, and behavior
`AIT_NATIVE_ACTOR`	**ait-core · user or operator · not secret.** Authorship identity recorded with local provenance and remote requests.
`AIT_NATIVE_SERVER_DATA`	**ait-core and ait-server · operator · not secret.** Durable server authority root used when `ait-server --data` is not supplied and by server-aware core operations. Keep it on durable storage.
`AIT_REPO_ROOT`	**ait-cli · user or supervisor · not secret.** Explicit repository root used for process discovery. Prefer entering the repository normally; use this only when the caller cannot establish the working directory.
`AIT_RUNTIME_DATA`	**ait-core · operator · not secret.** Runtime-data root shared by local diagnostics, Snapshot metadata, status caches, and server-aware operations.
`AIT_SERVER_TOKEN`	**ait-runner · operator · secret.** Optional bearer credential sent by the runner to the configured `ait-server` endpoint. Server URL and worker identity remain explicit runner command options.

Agent worker and integration credentials

Agent workers select a typed worker manifest first. These environment values supply the bounded worker bootstrap and credentials that cannot safely be embedded in ordinary repository content.

Name	Owner, setting, secret, and behavior
`AIT_AGENT_CONFIG_PATH`	**ait-agent-worker · user or supervisor · not secret.** Path to the typed Agent worker manifest. The ordinary repository default is `.ait/agent-workers.json`.
`AIT_DISCORD_APPLICATION_ID`	**ait-agent-worker · operator · not secret.** Discord application identity for the selected worker.
`AIT_DISCORD_BOT_TOKEN`	**ait-agent-worker · operator · secret.** Discord bot credential for the selected worker.
`AIT_DISCORD_PUBLIC_KEY`	**ait-agent-worker · operator · not secret.** Discord public verification key used to authenticate interaction requests.
`AIT_LINE_CHANNEL_ACCESS_TOKEN`	**ait-agent-worker · operator · secret.** LINE channel access credential for the selected worker.
`AIT_LINE_CHANNEL_SECRET`	**ait-agent-worker · operator · secret.** LINE request-verification secret for the selected worker.

Name	Owner, setting, secret, and behavior
`AIT_OPENAI_API_KEY`	**ait-agent-worker · operator · secret.** OpenAI credential made available to the selected worker when that worker uses the OpenAI boundary.
`AIT_SLACK_APP_TOKEN`	**ait-agent-worker · operator · secret.** Slack app-level credential for the selected worker.
`AIT_SLACK_SIGNING_SECRET`	**ait-agent-worker · operator · secret.** Slack request-signature verification secret.
`AIT_TELEGRAM_BOT_TOKEN`	**ait-agent-worker · operator · secret.** Telegram bot credential for the selected worker.
`AIT_TELEGRAM_OPENAI_API_KEY`	**ait-agent-worker · operator · secret.** Telegram-scoped OpenAI credential when that worker uses a separate key.
`AIT_TELEGRAM_WEBHOOK_SECRET`	**ait-agent-worker · operator · secret.** Telegram webhook verification secret.

Server CI memory admission

These three variables belong only to server-owned CI admission. Task-worktree RAM placement is repository configuration under `task_worktree.memory_root`, not process-environment configuration. Its typed fields, macOS 8 GiB default, read-only diagnostic, and persistent-disk fallback rules are documented in Parallel Task Isolation.

Name	Owner, setting, secret, and behavior
`AIT_NATIVE_SERVER_CI_RAM_MIN_AVAILABLE_BYTES`	**ait-server · operator · not secret.** Optional exact free-byte floor applied when admitting server CI onto the validated RAM root. Unset adds no floor beyond mount validation.
`AIT_NATIVE_SERVER_CI_RAM_RECLAIM_TARGET_BYTES`	**ait-server · operator · not secret.** Optional post-reclamation free-byte target. The effective target is never lower than the admission floor.
`AIT_NATIVE_SERVER_CI_RAM_ROOT`	**ait-server · operator · not secret.** Absolute, validated memory-backed host root for isolated server CI work. It must remain outside persistent server authority.

Storage and remote-operation tuning

These are advanced bounded controls. Invalid values use the documented fallback or fail validation at the owning boundary; they do not rewrite existing stored objects.

Name	Owner, setting, secret, and behavior
`AIT_OBJECT_PACK_CHUNK_MIB`	**ait-server · operator · not secret.** Positive object-pack write chunk size in MiB. Unset, invalid, or zero uses 8 MiB.
`AIT_REMOTE_MUTATION_RESPONSE_DEADLINE_SECONDS`	**ait-core · operator · not secret.** Remote mutation response deadline. Ordinary mutations default to 10 seconds and Task Land defaults to 30 seconds; a value at or below zero disables only this response deadline.
`AIT_REMOTE_MUTATION_SETTLE_POLL_SECONDS`	**ait-core · operator · not secret.** Reconciliation polling interval after a remote mutation. Default `0.25` seconds.
`AIT_REMOTE_MUTATION_SETTLE_WINDOW_SECONDS`	**ait-core · operator · not secret.** Post-mutation reconciliation window. Default `5` seconds.
`AIT_TREE_PACK_CHUNK_MIB`	**ait-server · operator · not secret.** Positive Tree-pack write chunk size in MiB. Unset, invalid, or zero uses 8 MiB.

Managed process boundaries

The values below are observable in child processes but are not ordinary configuration. AIT overwrites them with the exact paths or ownership token for the active operation.

Name	Owner, setting, secret, and behavior
`AIT_EXTERNAL_<NAME>_REPO_ROOT`	**ait-runner and ait-core · AIT-injected family · not secret.** Runner-created path to one locked external repository. `<NAME>` is nonempty normalized uppercase ASCII with digits or underscores; the core diagnostic uses the concrete `AIT_EXTERNAL_CORE_REPO_ROOT` member.
`AIT_RUNNER_ATTEMPT_ROOT`	**ait-runner · AIT-injected · not secret.** Exact runner-owned root of the current execution attempt. The runner's parent location is selected with `--attempt-root`; ambient values cannot redirect a child attempt.
`AIT_RUNNER_WORKSPACE`	**ait-runner · AIT-injected · not secret.** Exact materialized workspace for the current execution.
`AIT_WORKSPACE_LOCK_OWNER_TOKEN`	**ait-cli · AIT-injected · secret.** Opaque token propagated only to validated nested commands sharing the current workspace lock. Do not synthesize or log it.

Diagnostics and release-boundary automation

Name	Owner, setting, secret, and behavior
`AIT_PERFETTO_TRACE`	**ait-core and ait-server · user or operator · not secret.** Nonempty output path for opt-in Perfetto trace capture in a tracing-enabled process.
`AIT_SHARED_CARGO_TARGET_DIR`	**ait-release · automation · not secret.** Explicit shared Cargo target directory used by the supported native release smoke boundary. It is not a general runtime storage selector.

What is deliberately excluded

Release-manifest placeholders are argv template tokens resolved by the release adapter, not environment inputs. Workflow-local CI `env:` aliases, test fixtures, failure-injection controls, generated IDs, report labels, and values owned only by repository build or deployment scripts are likewise outside the installed runtime contract. They should be explained next to the workflow or script that owns them, not presented as settings users must understand.

This scope rule keeps the appendix complete without turning every source token into a public configuration promise.

Related

- Appendix: `.ait/config.json`
- Parallel Task Isolation
- ait-server: Remote Authority and Recovery

VERSION AUTHORITY

1.1.1 Source Authority

This PDF is a navigable public reference, not a separate product contract. Use the exact 1.1.1 source, Release, distribution contract, and CLI parser below when release authority matters.

PUBLIC SOURCE	https://github.com/weita2026/ait-native/tree/v1.1.1
PUBLIC COMMIT	`d248b29d3d988867366287d91e57b6c5bace716b`
RELEASE	https://github.com/weita2026/ait-native/releases/tag/v1.1.1
DISTRIBUTION SHA-256	`f2290e405520454e7926f09cdda6a1fe4be1b60300f03d4222918bedc9dd3efa`
CLI PARSER SHA-256	`829ccf69699c28bcca872cd2b191d256ed8aef4b4e8ada0590fb1e0aacb38215`
AIT-CORE	`SNP-ED7593DBF982`
AIT-SERVER	`SNP-0CCD7DD2A077`
AIT-RUNNER	`SNP-35C9C133D2EE`
AIT-PYTHON	`SNP-756C731A4CC0`
AIT-NODE	`SNP-55E90D0A81F1`

For the current online edition, search, and route-specific updates, open ait-native.dev/technical/.