Contributing to AIP

Use this guide to prepare a focused AIP code or documentation change whose implementation, tests, schemas, documentation, and evidence agree. A change is ready for review only when its claims are no broader than the exact artifacts and chec

SECTION	Project and Releases
TYPE	Project policy
PROTOCOL	AIP 1.0
SOURCE	CONTRIBUTING.md

This guide reflects AIP Core source revision `d7cce13d1d555644d04a4d73c66c95b113737635` and the staged documentation v2 workflow. It does not define hosting rules that are absent from source.

Choose the correct repository boundary

AIP Core and the documentation frontend are separate publication surfaces.

Change	Primary source boundary
Rust model, runtime, transport, profile, connector, host, CLI, schema generator, tests, or deployment code	AIP Core source repository
User-facing guides, reference, architecture, qualification reports, connector documentation, or navigation	Documentation repository
Behavior that changes both	Coordinated code and documentation changes, each reviewed in its owning repository

The AIP Core publication intentionally excludes the documentation frontend and most Markdown. Do not add frontend assets to the code repository or treat a documentation edit as proof that code changed.

The source-owned hosting signals are limited:

- GitHub CI runs for pull requests and pushes to `main`;
- Gitea CI runs for pull requests and pushes to `develop`;
- `CODEOWNERS` assigns every code path to `@getaip-core`.

These files do not establish branch protection, required approval count, merge strategy, issue workflow, response time, CLA, DCO, or signed-commit enforcement. Confirm the target branch and any hosting-only requirements with the repository maintainer before opening the change.

Complete the participation checks

Before writing code or documentation:

1. Read the code of conduct.
2. Send vulnerabilities through the private process in

SECURITY.md, never through a public contribution.

3. Review licensing and usage terms. The root

Business Source License states that an intentional contribution is submitted under its terms unless explicitly stated otherwise or covered by a separate agreement.

4. Confirm that you have authority to submit every source, test vector, schema,

example, fixture, image, and data sample in the change.

5. Keep the implementation clean-room. Do not copy third-party implementation

code or documentation without an explicit provenance and license review.

Never commit credentials, tokens, customer data, production identifiers, private qualification logs, local state, or generated secret files. Stop and use the private security process if sensitive material entered Git history.

Classify the change before editing

Change class	Artifacts that normally move together
Documentation only	One Markdown owner page, frontmatter, navigation, local links, source ledger, and applicable examples
Rust implementation	Source, public API documentation, focused tests, compatibility impact, and user-facing behavior documentation
Native wire or semantic model	Rust model, validation, generated native schemas, positive and negative conformance, affected profiles, normative specification, and release impact
Compatibility profile or transport	Profile DTO and mapping, transport framing, version negotiation, positive and negative interoperability tests, and profile documentation
Product connector	Operation mapping, support declaration, trusted context, credentials, host wiring, deterministic tests, conformance, product docs, and exact qualification scope
Deployment, persistence, or release	Configuration, migrations, least privilege, rollback and recovery, operator docs, artifacts, supply-chain evidence, and release notes

Split a change that has unrelated outcomes. A protocol correction, connector feature, dependency refresh, and prose cleanup should not share one review unless they form one inseparable contract change.

Preserve the architecture boundaries

Read the dependency graph before adding a crate edge.

- Native semantic types and validation belong in `aip-core`; product and

compatibility DTOs do not.

- Runtime owns lifecycle, durable state, policy enforcement, recovery, and

operational queries.

- Profiles translate external protocols without redefining native AIP

semantics.

- Connectors translate product behavior and expose only implemented,

support-declared operations.

- Transports frame and carry messages; authenticated gateway and runtime

boundaries make authorization decisions.

- Product-neutral `getaip-server` and fleet infrastructure must not depend on a product

connector.

- Each public product host owns exactly one of Cal.diy, Hermes Agent, Chatwoot,

Dify, CrewAI, or Twenty.

- Credentials remain host-owned and never enter protocol envelopes, generated

schemas, logs, or review examples.

Source-owned boundary checks reject product leakage into the core daemon, invalid facade features, and inconsistent private-SDK or connector-repository closures.

Follow a focused development workflow

1. Record the exact starting revision and verify that unrelated local changes

will remain untouched.

2. Write a one-sentence reader or operator outcome and list the affected

architecture, compatibility, security, storage, and deployment boundaries.

3. Create a focused branch from the confirmed target branch.
4. Add the smallest failing test or validation that demonstrates the intended

behavior, then implement the change.

5. Run focused formatting, lint, and tests while iterating.
6. Update every owned schema, compatibility mapping, guide, reference page,

and release-facing record required by the change class.

7. Run the applicable complete gates and prepare exact review evidence.

Use English for source, public APIs, comments, schema descriptions, documentation, and commit-facing engineering text. Keep commits reviewable and exclude generated runtime artifacts and unrelated formatting.

Keep generated schemas source-owned

Never edit a file under `schemas/aip` as the only implementation of a semantic change. Update the Rust model and validation, then run the source generator:

```
SH
cargo run -p xtask -- schema
git diff --exit-code -- schemas/aip
```

The second command must be clean after committing the intended generated changes. Add positive and negative conformance coverage and update the AIP 1.0 specification when the contract changes.

Backward-incompatible behavior requires an explicit versioning decision. Do not silently change a stable message family while leaving its version and schemas unchanged.

Apply the connector contract

Connector changes must preserve the callable boundary described by the connector contract:

- consume trusted server-created execution context;
- declare only behavior implemented by the connector;
- keep credential material redacted and outside serializable protocol data;
- define approval, idempotency, retry, cancellation, transaction, and

reconciliation behavior accurately;

- return typed, redacted failures and retain uncertain-outcome evidence;

- keep provider DTOs and paths inside the product connector;
- update standalone host construction and admission evidence when support

changes.

Follow Build a connector for a new public integration. Deterministic tests prove mappings and failure behavior; they do not prove a live product deployment. Live claims require the exact source, image, configuration, provider revision, time, and retained result defined by the testing and qualification index.

Select the applicable validation level

Run a narrow check first. For a Rust package, substitute its exact Cargo package name:

```
SH
PACKAGE="aip-core"
cargo fmt --all --check
cargo test -p "${PACKAGE}" --all-features
cargo clippy -p "${PACKAGE}" --all-targets --all-features -- -D warnings
```

Then choose every broader gate affected by the change.

Boundary	Source-owned gate or evidence
Generated native schemas	`cargo run -p xtask -- schema` plus a clean `schemas/aip` diff
Core and connector repository separation	`xtask` daemon and private-SDK boundary checks
Release-affecting source	`tools/release/check-release-artifacts.sh`
Minimum Rust version	CI check with Rust `1.88.0`
Native NATS	Dedicated NATS workflow job
PostgreSQL runtime, registry, least privilege, and scale	Dedicated PostgreSQL workflow job and retained scale result
CrewAI sidecar	Frozen Python environment, Ruff, and sidecar tests
Fleet images and failures	Migration-image, failure-matrix, and product-image qualification procedures
macOS portability	Selected protocol crates on the macOS workflow runner
Complete history secrets	Immutable Gitleaks workflow image with redacted output
Live external product	Product-specific isolated campaign and retained evidence

The release script is broad. It runs publication hygiene, formatting, architecture and repository boundaries, applicable semantic-version checks, schema generation, Clippy, all-feature Nextest, no-default tests, dependency policy, vulnerability audit, unused-dependency detection, fuzz compilation, and Rust documentation.

Do not report a gate as passed unless it ran successfully against the exact review commit. A skipped opt-in test, a workflow definition, or an older run is not current evidence.

Author documentation sequentially

Documentation follows the mandatory AIP documentation writing standard. Only one public Markdown document is active at a time. It must move through source audit, outline, complete draft, technical review, editorial review, formal validation, and acceptance before the next page starts.

Documentation changes must:

- cite exact code, schema, manifest, or retained evidence in the acceptance

record;

- use one reader outcome and one canonical owner for each fact;
- distinguish normative, implemented, conformant, qualified, live, historical,

planned, and excluded claims;

- keep examples fictional, safe, complete, and free of credentials or private

identifiers;

- preserve navigation order and resolve every local link;
- use only the six maintained public connectors;
- avoid publishing internal design-process and controlled-validation material.

The staged v2 Markdown tree is not an input to the current frontend build scripts. A passing frontend build therefore does not validate v2 content yet. Retain the writing-standard checks and cross-link evidence until the staged tree is wired into the site pipeline and its build becomes an additional gate.

Release-facing changes update `CHANGELOG` during release preparation. Do not invent a released version or date for an untagged source revision.

Prepare the review evidence

Use a compact summary with no unsupported claim:

```
TEXT
Problem and outcome:
Exact source revision:
Files and architecture boundaries:
Protocol and API compatibility:
Security and tenant isolation:
Storage, migration, recovery, and rollback:
Commands and tests run:
Retained evidence and artifact digests:
Documentation and generated schemas:
Known gaps, skipped cases, and non-claims:
```

Reviewers need the actual terminal state and artifact identity, not only a command list. Redact secrets and customer data before attaching logs. Keep raw qualification evidence in its protected retention boundary and publish only the reviewed redacted record.

Check merge readiness

A contribution is ready for owner review when:

- the stated outcome is implemented without unrelated changes;
- architecture, product-neutrality, tenant, credential, and generated-source

boundaries remain intact;

- every applicable focused and complete gate has exact evidence;
- schema, specification, compatibility, documentation, and release records

agree with code;

- migrations have verification, rollback, and uncertain-outcome handling;
- external-product and performance claims cite their exact retained campaigns;
- local links and navigation resolve and no internal-only material is exposed;
- remaining risks, skipped checks, and intentionally unsupported behavior are

explicit.

`CODEOWNERS` routes source review to `@getaip-core`; it does not replace the technical, editorial, security, release, or evidence review required by the change.

Related documentation

- Documentation home
- AIP 1.0 specification
- Documentation writing standard
- Code of conduct
- Security policy