Implementation status

Use this page to determine what the pinned AIP source implements and what its available evidence can establish. The key distinction is simple: source code, tests, and workflow definitions show that a behavior or check exists. They do not sh

SECTION	Qualification and Evidence
TYPE	Reference
PROTOCOL	AIP 1.0
SOURCE	docs/reference/implementation-status.md

This review covers GetAIP software `2.0.0` at source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. The separately retained release record proves the named protected source, scale, migration-image, failure, product-image, and controlled product-fleet gates. A later commit, rebuilt image, changed provider, or different deployment requires new evidence.

Read the status terms consistently

Term	Meaning on this page
Implemented	The pinned source contains the behavior and its public contract
Deterministic evidence present	A test or controlled harness exists in the pinned tree; an executed result is named separately when retained
Historical observation	A retained report records a run against the older artifacts named in that report
Qualified	An exact artifact and pinned dependencies passed a defined live campaign with retained evidence
Not established here	The pinned tree does not contain enough result evidence for the claim

Do not read `implemented` or `deterministic evidence present` as `PASS`. See conformance and qualification for the evidence ladder and claim vocabulary.

Identify the reviewed source

Property	Reviewed value
Source revision	`d7cce13d1d555644d04a4d73c66c95b113737635`
Release tag	`v2.0.0`
Workspace version	`2.0.0`
Native wire version	`1.0`
Rust packages	60 workspace members
Native schema registry	52 generated JSON Schema files
Maintained product connectors	Cal.diy, Hermes Agent, Chatwoot, Dify, CrewAI, and Twenty
Workspace publication setting	Every package has `publish = false`

The annotated `v2.0.0` tag and production marker resolve to the reviewed commit and tree. A retained clean clone-back matched those refs, Cargo metadata, the naming validator, and the protocol diff. This source identity does not by itself establish public registry publication, signatures, attestations, or a target deployment.

Status by implementation surface

Surface	What the pinned source implements	Evidence present in the tree	Remaining boundary
Native semantic core	Signed envelopes, identities, discovery, sessions, actions, delegation, events, approvals, transactions, callbacks, receipts, audit views, and validation	Typed models, generated schemas, fixtures, property tests, and core checks	Independent implementation interoperability and release-specific results
Trust and policy	Transport-bound principals, signer policy, authorization, replay protection, Ed25519 signatures, canonical JSON, session encryption, and redacted audit queries	Unit and property tests plus deterministic authority implementations	External security assessment, key operations, and target identity-provider qualification
Runtime and storage	Lifecycle execution, retry, cancellation, streaming, approvals, transactions, reconciliation, delegation, callbacks, receipts, leases, and recovery	In-memory, file, and PostgreSQL paths with deterministic and recovery tests	Target multi-node fault, backup, restore, disaster recovery, and capacity campaigns
Native HTTP	Manifest, message, action, query, receipt, event, SSE, and WebSocket-facing daemon behavior	Route, codec, authorization, cursor, follow, and process-boundary tests	Independent clients and the media-type difference documented below
Native NATS	Subject mapping, request/reply, listeners, queue groups, reconnect, and runtime routing	Codec and integration-oriented tests are present	Pinned broker failover, partition, throughput, and recovery evidence
MCP	Server and client roles, sessions, stdio, legacy HTTP with SSE, Streamable HTTP, replay, OAuth helpers, and four pinned protocol versions	Golden, negative, mapping, client, codec, server, and version-transport suites	Complete independent client/server qualification for the promoted artifacts
A2A compatibility	Agent Card projection and eleven task, subscription, and push-notification operations over three route aliases	Mapping, state, credential, signature, and daemon tests	A dedicated A2A conformance suite and independent peer campaign
Generic webhook profile	HMAC-SHA256 verification with a timestamp-skew check	Helper-level tests and connector-specific security assertions	Provider header mapping, replay behavior, delivery semantics, and live evidence
Connector fleet	Signed admission, durable registry, topology-aware routing, remote dispatch, standalone hosts, lifecycle control, leases, orchestration, and revocation	Unit, PostgreSQL, scale, migration-image, product-image, controlled product-fleet, and failure gates passed for the exact release	External-provider and target-deployment qualification
Operator CLI	Native calls and queries, receipts, connector administration and orchestration, core diagnostics, and MCP inspection and client checks	Command parsing and supporting implementation tests	The narrow diagnostics and reconciliation gap documented below
Release automation	Source gates and multi-platform image build, SBOM, provenance, signing, verification, and attestation steps	Six protected Gitea contexts, exact tags, bounded local image identities, and clone-back proof are retained	Public multi-platform digests, signatures, attestations, and registry publication

This matrix describes source coverage, not relative maturity. A surface with many tests may still require a small but decisive independent or live campaign.

Status of the six connectors

The source freezes six product catalogues against exact upstream revisions. Capability totals describe the admitted reference manifest, not every route in the upstream product.

Connector	Implemented reference surface	Deterministic evidence in the pinned tree	Qualification boundary
Cal.diy	81 capabilities across business profiles, availability, bookings, schedules, teams, routing, webhooks, conferencing, and OAuth clients	Dedicated 12-family frozen connector driver, unit and contract tests, ignored live test, controlled product-fleet path	The historical live report names older source and image artifacts; repeat it for the promoted artifact

Connector	Implemented reference surface	Deterministic evidence in the pinned tree	Qualification boundary
Hermes Agent	35 capabilities per endpoint for discovery, chat, responses, runs, approvals, sessions, schedules, operator calls, and delegation	Dedicated 12-family frozen connector driver, unit tests, and controlled product-fleet path	The historical two-endpoint report names older artifacts; repeat it with the target model, provider, and images
Chatwoot	153 capabilities: 150 pinned account operations and three composite conversation or message operations	Unit tests and controlled product-fleet execution with an upstream emulator	No complete retained isolated-live product campaign is present for this revision
Dify	74 capabilities in the reference app-plus-knowledge topology	Unit tests and controlled product-fleet execution with an upstream emulator	No complete retained isolated-live app and knowledge-service campaign is present
CrewAI	10 capabilities per crew, including run, status, replay, cancellation, batch, training, testing, knowledge, and memory reset	Rust tests, locked Python sidecar tests, a deterministic real-CrewAI registry, and controlled product-fleet execution	The controlled model is not a live model provider; retain an exact upstream and provider campaign separately
Twenty	22 core-record, metadata, OpenAPI, and webhook capabilities for one workspace	Rust tests, a standalone host image, and controlled product-fleet execution	No executable external Twenty campaign or result is present in the pinned tree

With one endpoint, crew, and Twenty workspace, the six reference catalogues contain 375 capabilities. The retained release evidence records a successful 45-case controlled product-fleet gate and exact local product-image identities, including Twenty. Controlled upstream behavior is not external-provider qualification.

The Cal.diy isolated-live record and Hermes Agent isolated-live record remain valid only for the exact older artifacts they identify. Neither result is inherited by revision `d7cce13d1d555644d04a4d73c66c95b113737635`.

Know what evidence is retained

Evidence class	Status for the reviewed revision
Implementation source	Present
Schemas, tests, harnesses, and workflow definitions	Present
Generated native schema files	Present
Protected CI job logs for this revision	Retained separately with SHA-256 digests
Controlled product-fleet result for this revision	Retained separately; 45-case matrix passed
External-provider result covering all six connectors	Not retained in the source tree
Immutable release image digests and signature verification	Not established by this review
Target deployment security, recovery, and capacity campaign	Not established by this review

CI workflows define PostgreSQL recovery, registry, lifecycle, scale, Python, secret-scan, fleet failure, migration-image, and product-image jobs. A workflow definition says what automation should execute. Only its retained result can say what did execute for an exact commit.

Track the current implementation gaps

Native HTTP media type

The native transport helper emits `application/aip+json`, which is the media type defined by AIP 1.0 for a complete native envelope. The daemon's Axum JSON routes and the CLI use `application/json`. JSON shape and media-type compliance must therefore be reported separately until those request and response paths are aligned.

CLI transaction reconciliation

`TransactionMode::Reconcile` and runtime reconciliation are implemented. The CLI `--transaction-mode` enum exposes execute, dry-run, plan, commit, compensate, and rollback-not-supported, but not reconcile. Use a typed SDK or a native envelope when an operator must submit that mode.

Narrow CLI conformance commands

`getaip conformance run` checks one built-in message body or one supplied envelope and does not fail merely because the printed `passed` field is false. `getaip mcp conformance` runs only four external-client checks. Neither command is an all-surfaces conformance gate. The exact safe usage is documented in conformance and qualification.

A2A conformance

A2A compatibility code and tests exist, but the pinned source has no dedicated A2A conformance suite or A2A conformance CLI. Do not convert internal profile tests into an independent-peer claim.

Product and release evidence

The controlled release gates and local image identities are retained, but they do not replace isolated-live provider campaigns. Public image publication, signing, SBOM, and attestation claims still require their own exact registry and artifact evidence.

Advance a status claim safely

To move a surface or connector from `implemented` to a stronger status:

1. select the exact source tree and produce immutable build identities;
2. run the complete applicable deterministic and conformance matrices;
3. assert expected suite IDs and counts, not only an aggregate boolean;
4. run independent-client or isolated-live scenarios where the claim needs

external behavior;

5. retain configuration class, dependency revisions, topology, timestamps,

redacted raw evidence, and cryptographic digests;

6. run deployment-specific security, restart, fault, backup, restore, and

capacity campaigns before using `production-ready`;

7. publish only the narrow status supported by the retained artifact.

Repeat the affected layers after any source, dependency, credential policy, topology, image, or configuration change.

Related documentation

- AIP 1.0
- Conformance and qualification
- Testing and evidence index
- Connector documentation
- Release artifacts