AIP specification suite

Use this index to choose the authoritative document for an AIP 1.0 behavior, wire shape, compatibility mapping, connector boundary, conformance claim, or implementation-status question. It is primarily for protocol implementers and reviewer

SECTION	Protocol Standard
TYPE	Documentation
PROTOCOL	AIP 1.0
SOURCE	docs/spec/README.md

The normative protocol version is `1.0`. Implementation details on these pages are reviewed against Rust workspace version `1.0.0` at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. A workspace release and a protocol version answer different questions.

Choose the artifact by question

Question	Read	Authority
What must an AIP 1.0 participant do?	AIP 1.0 normative specification	Normative behavioral contract
Which JSON members, types, and constraints are valid on the wire?	JSON Schemas	Normative machine-readable wire contract
How does MCP, A2A, a webhook, or a native binding map into AIP?	Compatibility profiles	Supporting implementation reference; the normative boundary remains in AIP 1.0
What must a product connector preserve and implement?	Connector contract	Supporting boundary for the normative protocol and reviewed Rust SDK
What evidence supports a conformance or qualification claim?	Conformance and qualification	Claim and evidence interpretation
What does the reviewed Rust source currently implement?	Implementation status	Informative source and evidence status

Start with the first row when behavior is disputed. Start with schemas when generating, accepting, or validating JSON. Use implementation and evidence pages only after the required behavior is clear.

Apply the two-part normative contract

An AIP 1.0 implementation must satisfy both:

1. the behavioral requirements in the normative specification; and
2. every applicable versioned JSON Schema.

Schema validation establishes wire shape. It cannot fully express authentication, replay decisions, authorization, legal state transitions, idempotency ownership, transaction consistency, route identity, or external side-effect rules. The reviewed implementation therefore applies both generated schema validation and semantic validation from the native model.

The reverse is also true: code that performs a sensible transition is not conforming if it accepts or emits a wire object that violates the applicable schema.

If a normative prose rule and a versioned schema disagree, treat the disagreement as a specification defect. Do not silently invent a third behavior. A conformance report should identify the exact conflicting artifact and rule until the suite is corrected.

Interpret each document's authority

Artifact class	It can establish	It cannot establish alone
Normative specification	Required semantics, invariants, states, errors, security, versioning, and conformance classes	Exact implementation support or deployment readiness
Versioned schemas	Accepted JSON structure, names, types, closed objects, and schema constraints	Cross-record policy or provider outcome
Profile reference	The reviewed projection and endpoint behavior for one foreign or transport surface	A parallel native lifecycle or universal profile support
Connector contract	The generic product boundary and reviewed SDK expectations	Qualification of a specific connector or provider
Conformance evidence	Results for named artifacts, classes, procedure, and environment	Production fitness outside that scope
Implementation status	Source presence and retained evidence at the named revision	A new protocol requirement or current live availability

Normative key words such as `MUST`, `SHOULD`, and `MAY` have their defined meaning only in the normative specification. Similar prose on an architecture, guide, or implementation page explains the reviewed implementation unless that page explicitly quotes and links the normative rule.

Keep five version axes separate

Version axis	Reviewed value or form	What it versions
Protocol	`1.0`	Native AIP semantic contract and `Envelope.aip_version`
Rust workspace release	`1.0.0`	Source packages and implementation artifacts
Manifest schema	`aip-manifest/v1`	Participant manifest shape and admission contract
Native message family	For example, `aip.core.v1.action`	One registered message namespace and major family
Compatibility profile	For example, `aip.mcp.compat.v1`	One bounded foreign-protocol or delivery mapping

Matching the workspace release does not permit an implementation to emit a different protocol version. Matching `aip_version` does not prove support for every message family, profile, connector, or conformance class.

Receivers use the envelope protocol version and registered message type together. Manifests and profile negotiation declare narrower support. A deployment must reject unsupported combinations rather than infer compatibility from a shared `v1` suffix.

Follow a reading path

Goal	Recommended order	Result
Implement native AIP	AIP 1.0 scope and terminology → envelope and registry → lifecycle sections → security → conformance → schemas	A behavioral model plus exact wire validation
Implement a compatibility edge	AIP 1.0 native lifecycle → compatibility section → profile reference → applicable native schemas	A bounded mapping into the same action and lifecycle model
Implement a connector	AIP 1.0 capabilities, actions, errors, and security → connector contract → applicable schemas	A product adapter that preserves native authority and lifecycle boundaries
Review an implementation claim	Normative requirement → conformance page → implementation status → named retained artifacts	A claim limited to its source revision, class, environment, and evidence
Diagnose an artifact conflict	Identify protocol, schema, message-family, profile, and workspace versions → compare the exact rule and generated artifact	A reproducible specification or implementation defect

The full normative specification is intentionally comprehensive. For normal application integration, start with the task-oriented guides instead. Return to this suite when exact behavior, interoperability, or conformance is the question.

Change the suite as one contract

A public protocol change is complete only when all affected owners agree:

- normative prose describes the new behavior and compatibility rule;
- native types and semantic validation implement the same invariant;
- versioned schemas accept exactly the intended wire shape;
- message and profile identifiers reflect any incompatible boundary;
- positive and negative conformance cases cover the rule; and
- examples and supporting references use the released artifact.

An incompatible change requires a new applicable version boundary. An additive change is compatible only when existing receivers can process or ignore it under the normative extension rules. A documentation edit cannot redefine the wire contract without the matching protocol and schema changes.

Exact generated inventories belong to source-owned registries and release artifacts. This index deliberately does not copy the complete message, schema, field, profile, or operation lists.

Read implementation and evidence claims narrowly

Source review can establish that a type, validator, profile mapping, transport, or connector path exists at one revision. Conformance can establish that named requirements passed a named procedure. Qualification can add topology, failure, scale, or external-product evidence for exact artifacts.

None of those claims automatically establishes current production readiness. Deployment identity, configuration, storage, credentials, network policy, connector version, provider behavior, and retained results remain part of the claim.

Specification entry points

- AIP 1.0 normative specification
- JSON Schemas
- Compatibility profiles

- Connector contract
- Conformance and qualification