Conformance and qualification

Use this reference to select the evidence layer that matches a claim, run the available check correctly, and report its result against an exact artifact. Conformance asks whether a named contract is satisfied. Qualification asks whether an

SECTION	Protocol Standard
TYPE	Reference
PROTOCOL	AIP 1.0
SOURCE	docs/reference/conformance.md

This page describes the conformance surfaces present in GetAIP `2.0.0` at source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. The protected source gate passed for that release, but this page does not extend the result to a later worktree, rebuilt image, deployment, connector, or external provider. See implementation status for current claims and known gaps.

Choose the evidence layer

Evidence is cumulative only when the same artifact and applicable scope pass every required layer.

Layer	It can establish	It cannot establish alone
Structural validation	One instance matches a schema and basic invariants	Authorization, lifecycle, or side effects
Unit or fixture test	One deterministic code path matches its assertions	Process, network, storage, or independent-peer behavior
Core conformance	Named native envelope, manifest, profile-binding, and capability-contract checks pass	Every protocol requirement or deployed route
Profile conformance	A named profile version, mapping, codec, or state machine passes its suite	Native policy, an independent client, or another version
Connector conformance	An exact manifest and driver pass required behavior families	Real provider availability unless the evidence source is isolated-live
Integration test	Selected repository components work together	Release identity or arbitrary deployment topology
Isolated-live qualification	An exact artifact interoperates with pinned real dependencies in one topology	Other versions, accounts, regions, load, or production policy
Deployment campaign	The target deployment passes defined security, fault, restart, and capacity scenarios	Future changes or untested conditions

`implemented`, `conforming`, `qualified`, and `production-ready` are not interchangeable. A higher layer must retain the lower-layer identities instead of referring to a mutable branch or product name.

Identify the subject before testing

Record these values before accepting a result:

Identity	Minimum value
Source	Full commit and clean or dirty state
Source tree	Deterministic tree or archive digest
Build	Package version plus binary, archive, or image digest

Identity	Minimum value
Contract	AIP version, profile ID and version, or connector manifest digest
Dependencies	Peer, provider, database, broker, and client revisions in scope
Topology	Processes, trust domains, storage class, network boundary, and replica count
Suite	Exact command or entry point, expected check IDs, and suite revision
Time	UTC start and finish

A result without subject identity can describe a past observation, but it cannot qualify a newly built artifact.

Use the core diagnostic correctly

The CLI exposes one narrow diagnostic:

```
SH
cargo run --locked -q -p getaip-cli -- conformance run \
  | jq -e '.passed == true'
```

Without `--envelope`, this creates one native `ManifestRequest` body and runs `core.message_body`. It is a smoke check for one built-in message, not the complete core or schema registry.

To inspect one envelope:

```
SH
cargo run --locked -q -p getaip-cli -- conformance run \
  --envelope path/to/envelope.json \
  | jq -e '.passed == true'
```

The file must first decode as a typed `Envelope`. The diagnostic then applies combined envelope schema and core validation and prints `name`, `passed`, and `detail`.

The command itself does not convert a printed `passed: false` into an error exit. Keep the `jq -e` assertion, or parse and assert the field in the calling tool. A JSON decode or command error does fail the process.

Embedded manifest report

The reusable gateway manifest report contains five checks:

Check ID	Scope
`discovery.manifest`	Nonempty manifest version and profile set; nonempty capability names; object input schemas
`discovery.profile_set`	No empty profile IDs and every caller-required profile present
`discovery.profile_bindings`	Required compatibility metadata for recognized advertised profiles
`enterprise.capability_contract`	Cross-field side-effect, execution, retry, approval, data, credential, transaction, and compensation rules
`profile.mcp.tools_list`	Manifest capabilities project to named MCP tools

These are implementation checks, not a complete enumeration of AIP 1.0. Manifest admission also compares callable capabilities with implementation-support declarations; see the connector contract.

Schema export and schema drift are separate build checks. Use JSON schemas for the exact 52-file registry and validation layers.

Evaluate connector conformance

The shared connector harness returns thirteen checks:

1. `connector.atomic_manifest_admission`; and

2. one `connector.behavior.*` check for each of twelve behavior families.

Every scenario is requested from the driver. A scenario may report not applicable only when the published manifest does not require it.

Scenario ID	Required when	Required assertions
`identity_credentials`	Any capability is advertised	`transport_actor_bound`, `tenant_isolated`, `credential_handle_opaque`
`schema_enforcement`	Any capability is advertised	`invalid_input_rejected`, `invalid_output_rejected`
`idempotency_duplicates`	Any contract requires idempotency	`duplicate_suppressed`, `collision_rejected`, `delivery_id_stable`
`retry_exhaustion`	Any contract supports retry	`retryable_error_preserved`, `backoff_observed`, `exhaustion_dead_lettered`
`cancellation_races`	Any contract supports cancellation	`before_dispatch_cancelled`, `in_flight_cancelled`, `late_cancel_preserved_terminal`
`streaming_backpressure`	Any contract supports streaming	`ordered_chunks`, `bounded_backpressure`, `terminal_chunk_unique`
`errors_uncertain_outcomes`	Any capability is advertised	`protocol_fields_preserved`, `uncertain_outcome_reconciled`, `error_secrets_redacted`
`approval_lifecycle`	Any contract requires approval	`authority_verified`, `quorum_enforced`, `evidence_persisted`, `resume_once`
`transaction_lifecycle`	Any transaction contract exists	`plan_before_commit`, `provider_operation_checkpointed`, `reconcile_before_retry`, `compensation_governed`
`audit_redaction`	Any capability is advertised	`receipts_emitted`, `audit_correlated`, `sensitive_fields_redacted`, `raw_secret_absent`
`webhook_security`	The generic webhook profile is advertised	`signature_verified`, `skew_rejected`, `replay_rejected`
`restart_reconnect`	Any capability is advertised	`state_recovered`, `duplicate_effect_prevented`, `reconnect_cursor_resumed`

The registry contains 38 named assertions. For an executed scenario, every required name must be present and true, and the evidence must contain at least one nonempty artifact ID. A missing assertion, false assertion, empty artifact list, or driver error fails that scenario.

The driver accepts two evidence-source classes:

- `deterministic_provider_double`, which records real connector requests,

retries, cancellation, failures, and effects under controlled behavior; or

- `isolated_live_provider`, which exercises a pinned real provider.

Static booleans returned without executing the connector are not valid release evidence. An isolated-live source can support qualification only when the exact provider, artifact, topology, configuration class, timestamps, and raw artifacts are also retained.

The harness is reusable; its existence does not mean every connector invokes it or passes every applicable scenario. Check the exact connector result in implementation status and its qualification page.

Evaluate MCP conformance

MCP has seven callable suite entry points. The aggregate rows below overlap; do not add them into one total.

Suite entry point	Checks at this revision	Scope
Golden fixtures	104	Every stable method in all four versions against the pinned official schema snapshot
Negative fixtures	110	104 method and parameter mutations plus six cross-cutting malformed-input checks
Profile mapping	8	Version matrices, initialize metadata, tool listing, and result projection
External client	4	Initialize, tool discovery, optional resource discovery, and AIP manifest projection
Transport codecs	3	stdio JSON-RPC, Streamable HTTP POST classification, and SSE round trip
Version-transport server matrix	56	Seven lifecycle checks for each of eight executable version-transport pairs
Server aggregate	61	Five base server checks plus the 56-check matrix

The first three MCP schema snapshots use JSON Schema Draft 7. The `2025-11-25` snapshot uses Draft 2020-12. Golden fixtures cover 24, 24, 25, and 31 stable methods respectively. Unstable methods are not golden fixtures.

CLI client check

Run the external-client subset against Streamable HTTP:

```
SH
cargo run --locked -q -p getaip-cli -- mcp conformance \
  --url http://127.0.0.1:18080/mcp
```

Or against a command-launched stdio server:

```
SH
cargo run --locked -q -p getaip-cli -- mcp conformance \
  --command ./target/release/getaip-server \
  --arg=--mcp-stdio \
  --arg=--service-id \
  --arg=agent:getaip:server:mcp-conformance
```

This command runs only the four external-client checks and returns an error when any of them fails. Optional resource discovery tolerates an unsupported or failed resource list, so it does not prove resource support. The command does not run golden, negative, profile, codec, server, or all-version matrix checks.

A full MCP claim must name the profile version, transport, peer role, suite entry points, expected check IDs, and exact client and server artifacts. Read compatibility profiles for the supported version and transport matrix.

Record uncovered profile boundaries

The reviewed source has no dedicated A2A conformance suite or A2A conformance CLI. A2A profile and daemon tests are implementation evidence, not an independent interoperability claim.

The generic webhook helper exposes one signature-and-skew check. The connector harness adds signature, skew, and replay assertions when the webhook profile is advertised. Neither check qualifies a provider's raw header mapping or delivery behavior without connector and live evidence.

Core envelope and manifest checks do not exercise every HTTP route, SSE or WebSocket recovery path, NATS broker behavior, storage backend, or independent client. There is no single command that establishes conformance for every AIP surface at this revision.

Run the code-owned release gate

The repository release gate is:

```
SH
cargo run -p xtask -- release-check
```

It executes these source-owned checks in order:

Gate	Behavior
Formatting	`cargo fmt --all --check`
Dependency boundaries	Product-neutral daemon/fleet boundary and private SDK closure checks
Semantic versioning	Per-package checks against an earlier stable tag when one exists
Schemas	Regenerate the committed AIP schema directory
Lint	Workspace, all targets, all features, warnings denied
Main tests	Workspace all-feature tests through `cargo nextest`
Minimal features	Workspace tests with no default features
Dependency policy	`cargo deny check`
Vulnerability audit	`cargo audit`
Unused dependencies	`cargo machete`
Fuzz targets	Compile the fuzz workspace
API documentation	Build workspace docs for all features without dependencies

The release workflow installs pinned QA tool versions, checks publication hygiene, runs this gate, requires no schema diff, and scans full Git history for secrets.

The gate does not start external products, prove connector credentials, run an isolated-live campaign, establish broker or database failover, or qualify a target deployment's load and fault limits. Attach those campaigns separately when the release claim requires them.

Interpret a report safely

Both report types compute success by asking whether every recorded check passed. An empty check list therefore also evaluates as successful. Every consumer should assert the expected suite identity, check count, and stable check IDs before accepting the aggregate status.

Use this interpretation order:

1. verify artifact, contract, suite, and dependency identities;
2. verify the expected checks are present exactly once;
3. reject failed, missing, duplicate, or unexpected required checks;
4. verify that not-applicable scenarios are permitted by the published

contract;

5. resolve every artifact ID to retained, immutable, redacted evidence;
6. record limitations and unexecuted external conditions;
7. publish only the narrow claim supported by those facts.

A green report cannot repair a stale manifest, an unpinned dependency, or a missing scenario.

Retain reproducible evidence

Retain a machine-readable report and raw artifacts without credentials. At minimum, include:

- the subject identities listed above;

- exact command arguments and relevant tool versions;
- configuration class and topology without secret values;
- scenario and check IDs with pass, fail, or not-applicable status;
- action, transaction, receipt, trace, and provider request IDs when safe;
- restart, replay, cancellation, approval, reconciliation, and provider

observations required by the campaign;

- start and finish timestamps;
- raw artifact paths, sizes, and cryptographic digests;
- known omissions, expected failures, and environmental limits.

Copy published results from the retained report rather than rewriting them manually. A byte-for-byte comparison between the published artifact and the retained source prevents accidental status drift.

Use precise claim language

Say	Only when
`schema-valid`	The named instance passed the stated schema and invariant check
`implemented`	The behavior exists in the identified source or artifact and deterministic tests cover it
`conforming`	The exact subject passed the named complete applicable conformance matrix
`qualified`	The exact artifact and pinned dependencies passed the defined qualification topology and campaign
`observed live`	A timestamped live run produced retained evidence, but broader qualification is not claimed
`production-ready for <target>`	That target's required security, operations, recovery, fault, and capacity campaign passed

Always append the scope: profile version, connector, topology, artifact, and date as applicable. Avoid `fully supported` or `all tests passed` when the statement omits the exact suite and excluded boundaries.

Related documentation

- Implementation status
- Testing and evidence index
- Compatibility profiles
- Connector contract