Compatibility profiles

Use this reference to select an implemented AIP profile, match its stable version and route surface, and identify which semantics remain native AIP responsibilities. It covers the three compatibility profiles and four native transport profi

SECTION	Protocol Standard
TYPE	Specification reference
PROTOCOL	AIP 1.0
SOURCE	docs/spec/compatibility-profiles.md

The normative compatibility boundary is defined by AIP 1.0. This page records the implementation at revision `97be86e9efedf07ecf1783b03800f683f107fb04`; it does not extend that normative contract or qualify a deployment.

Choose a profile

The profile identifier selects a defined mapping or binding. It does not name a connector, process, credential, or deployment.

Profile ID	Class	Select it when
`aip.native.http.v1`	Native request/response binding	A client exchanges native envelopes or uses the daemon's native HTTP views
`aip.native.nats.v1`	Native broker binding	A trusted deployment routes native envelopes through NATS subjects
`aip.sse.stream.v1`	Native server-stream codec	A server emits one native envelope per SSE data field
`aip.websocket.stream.v1`	Native bidirectional codec	Peers exchange one native envelope per WebSocket text frame
`aip.mcp.compat.v1`	MCP compatibility profile	An MCP client needs a projected view of AIP capabilities and lifecycle
`aip.a2a.compat.v1`	A2A compatibility profile	An A2A client needs Agent Card discovery and task-oriented operations
`aip.http.webhook.v1`	Generic webhook verification profile	A connector normalizes and verifies a signed HTTP webhook before mapping it

Native bindings retain the AIP envelope as their application message. Compatibility profiles translate a foreign protocol's DTOs, lifecycle, and errors at the edge. The profiles and connectors concept page explains this layer model; the sections below provide exact lookup tables for the implemented surface.

Shared profile boundary

Every profile adapter remains outside the native semantic core. A successful wire decode is only the first step:

1. the selected profile validates its own framing and version rules;
2. the adapter maps the request to a native message or bounded native query;
3. the gateway authenticates the transport-bound principal and applies

validation, replay, authorization, approval, and lifecycle policy;

4. the adapter projects the native result or typed error back to the foreign

protocol.

A profile cannot weaken native admission rules. A foreign request ID, tool name, task ID, header, or Agent Card field does not independently establish an AIP principal, tenant, capability grant, approval, idempotency reservation, or terminal result.

Profile support also has three separate meanings:

Claim	What it establishes
Identifier present in a manifest	The endpoint declares the profile
Mapping present in source	The reviewed implementation contains the DTO or transport mapping
Route enabled and qualified	An exact artifact and configuration passed the applicable checks

Do not infer the third claim from either of the first two.

MCP compatibility profile

`aip.mcp.compat.v1` maps MCP JSON-RPC requests to AIP capabilities, resources, actions, cancellations, event views, and typed results. MCP sessions, request correlation, roots, sampling, elicitation, prompts, completions, and task views remain profile state; they do not become native AIP message families.

Stable protocol versions

The implementation recognizes four stable MCP versions. The method set is versioned even when two versions currently have the same count.

MCP version	Stable methods	Change from the 24-method base
`2024-11-05`	24	Base set
`2025-03-26`	24	Same implemented stable method set
`2025-06-18`	25	Adds `elicitation/create`
`2025-11-25`	31	Adds elicitation completion and five task methods

The 24-method base is grouped here for exact lookup:

Group	Stable methods
Lifecycle and control	`initialize`, `notifications/initialized`, `ping`, `notifications/cancelled`, `notifications/progress`
Tools	`tools/list`, `tools/call`, `notifications/tools/list_changed`
Resources	`resources/list`, `resources/read`, `resources/templates/list`, `resources/subscribe`, `resources/unsubscribe`, `notifications/resources/list_changed`, `notifications/resources/updated`
Prompts	`prompts/list`, `prompts/get`, `notifications/prompts/list_changed`
Completion	`completion/complete`
Logging	`logging/setLevel`, `notifications/message`
Roots	`roots/list`, `notifications/roots/list_changed`
Sampling	`sampling/createMessage`

`2025-11-25` adds `notifications/elicitation/complete`, `tasks/list`, `tasks/get`, `tasks/result`, `tasks/cancel`, and `notifications/tasks/status`. Unstable parser-only methods are excluded from this matrix and are not part of the stable advertised surface.

MCP transports and routes

Transport choice constrains the executable version matrix:

Transport	2024-11-05	2025-03-26	2025-06-18	2025-11-25
stdio	Yes	Yes	Yes	Yes
Legacy HTTP and SSE	Yes	No	No	No
Streamable HTTP	No	Yes	Yes	Yes

The daemon exposes these network routes:

Surface	Method	Routes
Streamable HTTP	`GET`, `POST`, `DELETE`	`/mcp`, `/mcp/v1`, `/aip/v1/mcp`
Legacy SSE stream	`GET`	`/mcp/legacy/sse`
Legacy client messages	`POST`	`/mcp/legacy/messages`
Protected-resource metadata	`GET`	`/.well-known/oauth-protected-resource`, `/.well-known/oauth-protected-resource/mcp`

On Streamable HTTP, `POST` carries JSON-RPC messages, `GET` opens the server event stream, and `DELETE` removes the addressed session. The transport processes `MCP-Protocol-Version`, `MCP-Session-Id`, and `Last-Event-ID`. Authentication, allowed origins, request bounds, session ownership, and resume behavior remain deployment and transport controls.

stdio carries one JSON-RPC frame per line and relies on the spawning host for process identity and isolation. It has no HTTP bearer, Origin, or session-header boundary.

MCP projection and stable facade

`tools/call` becomes a native `Action` and follows the same gateway and runtime path as a native request. Request-to-action correlation allows an MCP cancellation notification to target the active action. The result is projected back as MCP content, structured content, a task view when enabled, or a JSON-RPC error derived from the native error.

The server can expose generated tools for a bounded local manifest and stable AIP facade tools. The stable surface includes `aip_capabilities` for discovery and `aip_call` for invocation, plus focused lifecycle, event, approval, resource, receipt, transaction, and delegation tools.

When a tenant-scoped connector catalog is installed, generated fleet capabilities are not expanded into one global in-memory `tools/list`. `aip_capabilities` queries the catalog using the identity bound to the MCP session and returns a bounded page with `catalog_revision`, `total`, `capabilities`, and `next_cursor`. This preserves tenant visibility and revision-fenced pagination while keeping the MCP tool surface stable.

Server features remain composition-dependent. A method may belong to the negotiated MCP version while its provider is unavailable, disabled, or empty. For example, roots, sampling, elicitation, dynamic notifications, tasks, and generated capability tools are advertised only when the active composition can serve the corresponding behavior.

Use AIP through MCP for a task-oriented connection and invocation flow.

A2A compatibility profile

`aip.a2a.compat.v1` maps Agent Card discovery, messages, tasks, cancellation, streaming, and push-notification configuration to native AIP manifests, actions, results, events, and durable profile state.

Current A2A operations

The current advertised surface has eleven operation names:

Group	Operations
Send	`SendMessage`, `SendStreamingMessage`
Read and follow tasks	`GetTask`, `ListTasks`, `SubscribeToTask`
Cancel	`CancelTask`
Push configuration	`CreateTaskPushNotificationConfig`, `GetTaskPushNotificationConfig`, `ListTaskPushNotificationConfigs`, `DeleteTaskPushNotificationConfig`
Extended discovery	`GetExtendedAgentCard`

Legacy operation aliases may be accepted as inputs for compatibility, but they are not advertised as the current surface.

A2A routes

Surface	Method	Routes
Agent Card	`GET`	`/.well-known/agent-card.json`, `/.well-known/agent.json`, `/a2a/agent-card`
JSON-RPC	`POST`	`/a2a`, `/a2a/v1`, `/aip/v1/a2a`

The Agent Card can project manifest identity, documentation, skills, media modes, interfaces, extensions, security schemes and requirements, signatures, and supported streaming, push, or extended-card features. The advertised feature set must match the active composition.

`SendMessage` and `SendStreamingMessage` create native actions. `CancelTask` creates a native cancellation. Task status and artifacts project from the durable action view, while subscriptions project the event stream. The sender's message ID produces the native idempotency key `a2a-message:{message_id}`. Clients must reuse the message ID when retrying an uncertain send.

Push configurations and delivery cursors are profile state, not fields copied into the action. The daemon implementation encrypts stored push credentials, signs outbound payloads, validates callback destinations against egress policy, and uses leased retryable delivery. Those controls describe the reviewed daemon, not every independent implementation of the profile.

Use AIP through A2A for a complete discovery, send, stream, and recovery flow.

Generic HTTP webhook profile

`aip.http.webhook.v1` defines a small provider-neutral signing and verification boundary. It normalizes five values:

Field	Meaning
`delivery`	Delivery identity supplied to the signing input
`timestamp`	Unix time in seconds
`signature`	Standard-base64 HMAC-SHA256 bytes
`source_system`	Normalized source identifier
`event_type`	Normalized source event type

The signing input is the UTF-8 text `{timestamp}.{delivery}.{payload}`. Verification:

1. rejects a timestamp whose absolute distance from current UTC time exceeds

the caller-supplied skew;

2. recomputes HMAC-SHA256 with the configured secret;
3. decodes the expected and supplied signatures from standard base64;
4. rejects unequal lengths; and
5. compares equal-length signature bytes in constant time.

The generic profile does not define raw HTTP header names, an ingress route, payload schema, replay store, delivery deduplication, provider parser, or native message mapping. A connector owns those choices. It should verify the original request body before parsing, persist or reject repeated delivery IDs according to its contract, and only then construct the appropriate native AIP message.

Because the signing helper treats the payload as UTF-8 text, a connector that accepts arbitrary binary request bodies needs an explicitly defined binary canonicalization instead of assuming this profile covers it.

Native transport profiles

The four native profiles share the profile namespace but do not translate a foreign semantic model:

Profile	Implemented framing	Deployed daemon surface
`aip.native.http.v1`	Complete envelope encoded as JSON	Native message, action, manifest, and operational HTTP routes
`aip.native.nats.v1`	JSON transport wrapper, with raw-envelope receive compatibility	Structured subjects, publish, and request/reply
`aip.sse.stream.v1`	Complete envelope in each SSE `data` field	Durable event and action follow routes use a daemon-specific adapter
`aip.websocket.stream.v1`	Complete envelope in each text frame	`GET /aip/v1/ws` adds envelope exchange and subscription control

The framework-neutral SSE and WebSocket codecs use in-memory correlation or session queues. Their cursors and frames are not the durable event query shape of the daemon adapters. NATS Core delivery does not imply JetStream retention, and HTTP reachability does not establish peer identity.

Use transport bindings for exact native routes, media behavior, subjects, headers, framing, authentication, recovery, and implementation differences.

Negotiation and failure behavior

Profile negotiation must select an identifier and version supported by both peers and executable on the chosen transport. A manifest declaration does not override a transport-version restriction.

Failure	Interpretation
Unknown profile ID	No mapping is selected; reject or negotiate a supported profile
Unsupported stable version	Do not silently use version-specific methods from another matrix
Known method with unavailable provider	The protocol surface exists, but the active composition cannot serve it
Mapping or schema error	Reject before native dispatch when no valid native request can be formed
Native policy or authorization error	Project the typed native failure; do not reinterpret it as transport success
Connection loss after dispatch	Preserve the original mutation identity and query durable native state
Cursor rejected or expired	Re-establish the stream from a supported durable position and reconcile state

Compatibility does not provide exactly-once external effects. The native capability contract, idempotency key, action status, transaction state, receipts, and provider reconciliation remain authoritative after an uncertain outcome.

Conformance and evidence

Profile unit tests establish deterministic DTO and mapping behavior for the reviewed source. Transport and server tests cover additional lifecycle and route behavior. Neither proves that an arbitrary deployment has enabled the same providers, identity configuration, durable stores, callbacks, or external systems.

A conformance claim must name the profile, protocol version, transport, implementation revision, configuration boundary, suite, and retained result. Provider or connector qualification is a separate evidence layer. See conformance and qualification before publishing a compatibility or readiness claim.

Related documentation

- AIP 1.0 normative specification
- Profiles and connectors
- Transport bindings
- Conformance and qualification