Connector contract

Use this reference to implement or review a product connector without recreating AIP governance at the provider edge. It identifies the callable surface, the trusted context a connector receives, the behavior it may advertise, and the evide

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

This is a supporting implementation reference for source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. Native action, capability, transaction, error, and envelope semantics remain normative in AIP 1.0. Provider-specific operations belong in the applicable connector reference.

Contract boundary

A connector maps one product edge into the admitted AIP capability contract. It does not become a second gateway, policy engine, or action store.

Connector owns	AIP runtime owns
Provider request and response DTOs	Native action and session lifecycle
Provider authentication at the downstream edge	Transport-authenticated actor and tenant
Capability-to-provider operation mapping	Capability admission and input validation
Provider error normalization	Authorization, approval, and retry policy
Provider operation references and reconciliation	Idempotency reservation and transaction state
Verified provider events and channel replies	Durable events, results, receipts, and callbacks
Product-specific readiness detail	Fleet routing and client-facing readiness

The connector consumes trusted runtime context. It does not accept a tenant, principal, credential, approval, route, retry grant, or transaction state from ordinary action input or untrusted metadata.

Select the correct surface

The base `Connector` interface has four responsibilities:

Method	Contract
`id`	Return one stable connector identifier
`discover`	Return the connector manifest for the supplied connector context
`map_error`	Convert a connector error to a typed AIP protocol error
`health`	Report whether the connector can currently discover and serve traffic

The default health check treats successful discovery as ready. A connector with stronger dependencies should override it and return operator-safe detail. Readiness does not prove a provider operation, webhook, or credential has been qualified.

Five role-specific interfaces cover focused capability, event, channel, outbound, and escalation surfaces:

Interface	Additional responsibility
`CapabilityProviderConnector`	Return executable capabilities independently of transport ingress
`InboundConnector`	Map one external event payload to native envelopes
`ChannelConnector`	Ingest channel events and emit results to a conversation surface
`OutboundConnector`	Invoke an external system, observe cancellation, request remote cancellation, and emit a result
`EscalationConnector`	Map a governed human escalation to a product workflow

Callable production capabilities use `FrozenConnector`. That boundary receives a complete trusted execution context and makes every optional lifecycle operation explicit. A connector may also implement an event or channel interface, but those roles do not make its callable claims true.

Declare implementation support

For every admitted capability, `implementation_support` returns nine independent flags:

Flag	True means
`invocation`	Normal invocation is implemented
`cancellation`	Runtime cancellation reaches a downstream operation
`streaming`	Incremental output is implemented
`retry`	Retry classification and downstream idempotency are implemented
`transaction`	Transaction planning and commit behavior are implemented
`reconciliation`	An uncertain provider outcome can be reconciled
`compensation`	Compensation is implemented as a governed action
`approval`	Verified approval evidence and resume behavior are implemented
`credentials`	Credential handles resolve through the deployment provider

Manifest admission compares these flags with the capability contract:

Advertised contract behavior	Required implementation flag
Any callable capability	`invocation`
`execution.supports_cancel`	`cancellation`
`execution.supports_streaming`	`streaming`
`execution.supports_retry`	`retry`
A transaction contract	`transaction`
Transaction mode `reconcile`	`reconciliation`
Supported compensation	`compensation`
Required approval	`approval`
Required credentials	`credentials`

A missing required claim produces an admission issue. A true flag is still an implementation declaration, not conformance or provider qualification. Resources are not required to declare callable invocation.

Consume trusted execution context

`ActionExecutionContext` is deliberately non-serializable. The runtime constructs it from authenticated and durable state for each attempt.

Field	Connector-visible authority or facility
`actor`	Transport-authenticated principal, issuer, scheme, scopes, and lifetime
`tenant`	Verified tenant membership when the action is tenant-scoped
`credential`	Opaque credential handle, not secret bytes
`deadline`	Absolute execution deadline and remaining-time calculation
`cancellation`	Cooperative cancellation signal
`idempotency`	Reservation owned by this attempt when applicable
`approval`	Verified approval IDs, decision IDs, policy hashes, and authorization
`transaction`	Transaction ID, provider operation ID, and reconciliation cursor
`transaction_checkpoint`	Durable provider-operation checkpoint publisher
`execution_checkpoints`	Provider-effect durability checkpoint publisher
`stream`	Runtime-owned incremental chunk publisher
`trace`	Trusted trace and span identifiers
`redaction`	Input, output, and JSON-pointer redaction policy

The context is attempt-scoped. Do not serialize it, place it in a provider payload, retain it as a credential cache, or reconstruct it from fields the caller controls. A remote connector host derives the equivalent context only after verifying the signed, route-pinned central request.

Implement typed operations

`FrozenConnector` requires an implementation-support declaration and `invoke_typed`. Seven other operations default to `connector.operation_unsupported`, category `permanent`, with `retryable=false`.

Method	Purpose
`invoke_typed`	Execute a normal capability action
`plan_typed`	Plan or dry-run a governed mutation
`commit_typed`	Commit a prepared mutation
`compensate_typed`	Execute a separately governed compensation
`cancel_typed`	Request cancellation of a submitted downstream operation
`reconcile_typed`	Determine the terminal outcome of an uncertain provider operation
`emit_typed`	Emit a terminal result or provider event
`ingest_typed`	Map one provider event under trusted context

Failures identify one of twelve serialized `ConnectorOperation` values:

Group	Operation values
Discovery and readiness	`discovery`, `admission`, `health`
Provider execution	`invocation`, `cancellation`, `streaming`
Transaction recovery	`transaction_plan`, `transaction_commit`, `reconciliation`, `compensation`
Provider data exchange	`emission`, `ingestion`

The runtime adapter rejects an action whose capability ID differs from the capability bound to the handler. It selects the method from transaction mode:

Action transaction mode	Selected connector method
None, `execute`, or `rollback_not_supported`	`invoke_typed`

Action transaction mode	Selected connector method
`dry_run` or `plan`	`plan_typed`
`commit`	`commit_typed`
`compensate`	`compensate_typed`
`reconcile`	`reconcile_typed`

Reconciliation requires transaction context and a durable provider operation ID. A nonterminal reconciliation result becomes `transaction.reconciliation_pending`, category `temporary`, retryable after 5,000 ms, with its cursor and redacted evidence retained. A terminal result records whether the original commit completed.

Cancellation is separate from transaction dispatch. The runtime signals the cooperative token and can call `cancel_typed` with the same trusted context. Dropping a local future does not prove that a remote provider stopped work.

Resolve credentials at the provider edge

The execution context carries only an opaque `CredentialHandle`. The connector's deployment credential provider resolves that handle inside the connector process and for the verified tenant and account.

`ConnectorSecret` is the in-memory wrapper available to connector implementations. It is intentionally non-serializable, renders as `[REDACTED]` in diagnostics, compares equal-length material in constant time, and zeroizes its bytes on drop. Secret bytes or UTF-8 text should be exposed only while constructing the downstream request.

Do not copy credential material into:

- manifests or capability schemas;
- action input, metadata, or trace fields;
- protocol errors or redacted details;
- audit events, receipts, metrics, or logs;
- provider operation references or reconciliation cursors.

Empty or unresolvable credential material is a typed connector failure. It is not a reason to fall back to caller-supplied credentials.

Stream, cancel, and honor deadlines

For a streaming capability, publish each `StreamChunk` through `context.stream.emit`. The runtime-owned publisher persists and exposes the chunk through the action lifecycle. A connector should not call the client-supplied callback directly or maintain a second authoritative stream cursor.

Check `context.cancellation` before a downstream side effect, while waiting between provider events, and before expensive follow-up work. If the provider supports cancellation, `cancel_typed` should use the retained provider operation reference. Otherwise advertise `cancellation=false` and let the unsupported result remain explicit.

Use `context.deadline` to bound downstream requests. The runtime also applies the effective action timeout. On timeout it signals cancellation, attempts connector-specific cancellation, and returns `sla.timeout_exceeded`. Timeout retryability is true only when the admitted capability supports retry and the action is not a commit or execute transaction.

Mark durability boundaries

Two checkpoint publishers solve different recovery problems:

Publisher	Call it when	Meaning
`transaction_checkpoint.checkpoint`	A provider operation ID exists, before awaiting a commit response that may be lost	The runtime can reconcile the exact provider operation
`execution_checkpoints.provider_effect_committed`	The provider system of record has durably accepted an idempotent effect, before returning its response	A bounded observer can distinguish a pre-effect crash from a post-effect crash

The transaction checkpoint accepts a `ProviderOperationRef` and optional opaque reconciliation cursor. It is unavailable outside a transactional action. The provider-effect checkpoint is a process-local notification; any retention is the installed observer's responsibility. It carries no provider payload or credentials.

These calls do not settle the native action. The connector must still return a typed result or failure, and the runtime persists terminal lifecycle and idempotency state.

Return typed results and failures

A successful operation returns an `ActionResult` for the supplied action. Output and message parts must match the admitted capability contract and redaction policy. Streaming output does not replace the terminal result.

`ConnectorFailure` preserves twelve fields:

Field	Contract
`code`	Stable namespaced failure code
`message`	Human-readable redacted summary
`category`	AIP error category
`retryable`	Whether retry is safe under the capability's idempotency contract
`retry_after_ms`	Optional provider-suggested delay
`provider_request_id`	Optional provider request or trace ID
`provider_operation`	Optional durable operation reference
`remote_status`	Optional provider protocol status, normally HTTP
`uncertain_outcome`	Whether an external effect may have occurred
`redacted_details`	Structured details safe for durable audit storage
`source`	Connector or provider component that produced the failure
`operation`	One of the twelve typed connector operations

Conversion to `ProtocolError` retains retry, provider, remote-status, uncertain-outcome, redacted-detail, and source information. It does not make an unsafe retry safe.

When `uncertain_outcome=true`, return every known provider request and operation reference, avoid inventing a terminal result, and use `reconcile_typed` when the capability declares reconciliation. A client or runtime should reuse the original action and idempotency identity. Starting a new mutation can duplicate an external effect.

Preserve the contract across placement

The product contract is the same for local and remote placement:

Placement	Additional boundary
Trusted local module	Connector code, credentials, and resource use share the gateway process
Remote connector host	A signed native request pins tenant, instance, replica, version, manifest, lease, credential revision, and capability before context construction

Remote placement adds route, peer, capacity, callback, and lease checks. It does not authorize the connector to trust action metadata or change capability semantics. See the connector fleet HTTP API for that wire boundary.

Review an implementation

Before admitting a connector capability, confirm all of these points:

- the manifest describes only implemented operations and schemas;
- every callable capability has an exact nine-flag support record;
- the connector uses only trusted actor, tenant, credential, approval,

transaction, and idempotency context;

- secret material is resolved at the provider edge and excluded from durable

output;

- retries use the original mutation identity and match the provider's

idempotency behavior;

- streaming, cancellation, planning, commit, compensation, and reconciliation

are either implemented or explicitly unsupported;

- provider operation IDs are checkpointed before an uncertain commit wait;
- provider-visible durable effects publish the effect checkpoint at the

documented boundary;

- failures preserve redacted recovery evidence and identify uncertain

outcomes;

- local or remote placement retains the same capability and lifecycle meaning.

This review establishes contract alignment for the inspected implementation. Conformance and live provider qualification require separate suites and retained evidence.

Related documentation

- Capabilities and contracts
- Profiles, transports, and connectors
- Build a connector
- Conformance and qualification