Errors and retry decisions

AIP errors separate machine decisions from diagnostic text. Read the protocol code, category, retry hint, action or transaction state, and connector outcome evidence before deciding to correct, authorize, retry, reconcile, or stop.

SECTION	API Reference
TYPE	Reference
PROTOCOL	AIP 1.0
SOURCE	docs/reference/errors.md

This reference describes source revision `d7cce13d1d555644d04a4d73c66c95b113737635`. It covers native AIP and the product-neutral connector fleet. Product-specific error codes and recovery procedures belong to each of the six connector troubleshooting sections.

Protocol error object

`ProtocolError` has three required fields and four optional fields:

Field	Required	Meaning
`code`	Yes	Stable namespaced machine code
`message`	Yes	Human-readable diagnostic summary
`category`	Yes	One of the seven cross-implementation categories
`retryable`	No	Explicit statement about retry safety for this failure
`retry_after_ms`	No	Suggested delay before another eligible attempt
`details`	No	Structured context whose shape belongs to the error producer
`source`	No	Connector, profile, transport, or component metadata

Optional fields are omitted when absent; the Rust serializer does not emit them as `null`. Automation should use `code`, `category`, and explicit retry state. It must not match on `message` text.

```
JSON
{
  "code": "idempotency.in_progress",
  "message": "another execution owns this idempotency key",
  "category": "temporary",
  "retryable": true,
  "retry_after_ms": 1250,
  "details": {
    "key": "booking-create-0190",
    "owner_action_id": "act_0190c42f2d5a70008000000000000001",
    "reservation_expires_at": "2026-07-26T12:00:01.25Z"
  },
  "source": {
    "component": "aip-runtime"
  }
}
```

The protocol type permits arbitrary JSON in `details` and `source`. The typed connector boundary requires redacted details, but consumers must not assume that every producer enforced redaction. Apply authorization and logging policy before returning, retaining, or exporting either field.

Categories and native HTTP mapping

The native HTTP binding maps a protocol category to a recommended status:

Category	Native HTTP status	Meaning	Default operator decision
`temporary`	503	The component may recover without changing the request	Retry only through an eligible retry policy
`permanent`	400	The unchanged request cannot succeed	Correct input or stop
`auth`	401	Authentication or authorization failed	Re-establish identity, tenant, scope, or trust; do not broaden authority automatically
`policy`	403	An authenticated operation violates policy	Obtain the required decision or change the operation
`economic`	402	Billing or settlement state prevents execution	Resolve account or settlement state before another attempt
`connector`	502	Connector or downstream execution failed	Inspect connector evidence and outcome certainty
`transport`	503	The delivery path failed	Recover delivery, then determine whether execution may already have started

This mapping applies when an HTTP handler delegates status selection to the native binding. A route may intentionally use a more specific status. For example, connector-host capacity returns 429 and replay returns 409. Always parse the response body as well as the status.

Where an error appears

Native error envelope

A message-level failure is carried in an envelope with message type `aip.core.v1.error` and `body.error.error` containing `ProtocolError`. Native HTTP returns that envelope with a mapped status. Native NATS returns the same error envelope without an HTTP status.

The signed connector-host path correlates an error response to the request, addresses it to the request sender, and signs it. A client must verify the response signature and correlation before trusting the embedded error.

Failed action result

A capability may return an `ActionResult` whose `status` is `failed` and whose `error` is a `ProtocolError`. The enclosing HTTP request can still succeed. Inspect the action result instead of treating HTTP success as action success.

An acknowledgement can also reject or queue an action and carry `reason` and `retry_after_ms` without a `ProtocolError`. Treat those as acknowledgement state, not as a substitute error object.

Compact HTTP error

Some pre-envelope or ergonomic routes return a compact object under `error`. Examples include JSON decoding before an envelope exists and capability-catalog discovery. A compact error can omit category, retry fields, or source. Use the route's documented status and code; do not synthesize missing retry authority.

Model Context Protocol mapping

The Model Context Protocol (MCP) profile maps AIP categories to JSON-RPC codes:

AIP category	JSON-RPC code
`auth`	`-32001`

AIP category	JSON-RPC code
`policy`	`-32003`
`temporary`, `transport`	`-32010`
`connector`	`-32020`
`economic`	`-32030`
`permanent`	`-32602`

When an AIP error has `details`, the current mapper uses those details directly as JSON-RPC `data`. When details are absent, `data` contains AIP extension metadata with code, category, retry fields, and source. A generic MCP client must therefore not assume that every projected failure exposes the AIP code.

Retry decision order

Apply these checks in order. A later `yes` cannot override an earlier stop.

Check	Continue only when	Otherwise
Outcome certainty	No side effect occurred, or the same effect can be reconciled safely	Read durable action or transaction state and reconcile
Capability contract	`supports_retry` is true	Stop or follow the capability's recovery procedure
Retry safety	Safety is `safe`, or `safe_with_idempotency_key` with a key present	Do not repeat the action
Result state	The action result is `failed`	Follow the lifecycle for queued, pending, cancelled, or terminal success state
Explicit error hint	`retryable` is not `false`	Stop; correct the cause instead
Category fallback	`retryable` is `true`, or absent and the policy accepts the category	Stop unless a higher-level contract explicitly provides recovery
Attempt and elapsed budget	Another bounded attempt fits both limits	Dead-letter, reconcile, or escalate
Delay	Backoff and any retry hint fit the remaining budget	Stop or wait for a later operator decision

At this revision, queued-runtime retry policy behaves as follows:

- `safe` and `safe_with_idempotency_key` capabilities receive a maximum of

three attempts by default; `unsafe` and `unknown` receive one;

- `safe_with_idempotency_key` cannot retry without an idempotency key;
- `retryable: false` always stops the runtime retry;
- `retryable: true` permits the error-hint gate, subject to every other gate;
- absent `retryable` falls back to configured categories, whose default set is

`temporary`, `transport`, and `connector`;

- `retry_after_ms` is clamped to at least 1 ms and at most the policy's maximum

backoff; absent delay uses bounded exponential backoff;

- the retry is rejected if the next delay exceeds `max_elapsed_ms`.

These are current runtime defaults, not a protocol mandate for every AIP implementation. A client should read durable status before resubmitting after an interrupted transport exchange.

Unknown outcomes

A transport failure after dispatch does not prove that a provider mutation did not happen. A typed `ConnectorFailure` carries the evidence needed to choose a safe branch:

Connector field	Recovery use
`provider_request_id`	Correlate provider logs and support evidence
`provider_operation`	Identify provider, operation ID, and optional request ID
`remote_status`	Record the downstream protocol status
`uncertain_outcome`	Prevent blind replay when an external effect may exist
`operation`	Identify discovery, admission, invocation, cancellation, streaming, health, transaction, reconciliation, compensation, emission, or ingestion
`redacted_details`	Retain bounded diagnostic evidence without raw provider data
`source`	Identify the connector component that produced the failure

The conversion to `ProtocolError` places these values in `details` and preserves the connector's code, category, retry flag, and delay. Conversion in the other direction treats absent `retryable` as false.

For transaction modes `commit` and `execute`, the runtime moves the transaction toward `outcome_unknown` when the result has code `sla.timeout_exceeded` or `details.uncertain_outcome` is true. Use the captured provider operation ID and the reconciliation action. Do not create a second mutation merely because a transport or timeout category looks temporary.

Common native code families

The table lists product-neutral families implemented at the pinned revision. It is a lookup aid, not an exhaustive registry of provider-specific codes.

Code or prefix	Typical meaning	Decision
`action.invalid_input`	Required action data or a semantic invariant is invalid	Correct the action; unchanged retry is false
`capability.unsupported`	The endpoint or manifest does not own the requested capability or message	Rediscover capabilities or choose another endpoint
`resource.not_found`, `receipt.not_found`	Selected session, capability, resource, or receipt is absent	Check the identifier and tenant boundary
`handshake.no_compatible_profile`	No requested profile is supported	Negotiate a profile present in the manifest
`auth.native_http_unauthorized`	Native bearer authentication is missing, invalid, or not configured for the presented token	Correct authentication; do not retry with broader scopes
`auth.*`, `credential.*`, `audit.not_authorized`	Actor, claim, tenant, session, credential, ownership, or scope does not match trusted context	Re-establish the intended identity and authority
`auth.signature.invalid`, `replay.message_id`	Signature verification or replay fencing failed	Stop; correct signing, clock, message identity, or replay state
`query.invalid`, `query.invalid_cursor`	Query syntax or cursor format is invalid	Correct the query or restart pagination as documented
`stream.cursor_expired`	Cursor is outside the retained replay window	Read current durable state, then restart from an available cursor
`idempotency.in_progress`	Another action owns the scoped key until its reservation expires	Read the owning action; wait for the bounded retry delay
`policy.human_approval_required`	Execution is pending an approval lifecycle decision	Submit or await approval; do not retry the action
`sla.timeout_exceeded`	The handler exceeded its effective action timeout	Inspect cancellation and outcome certainty before any retry
`sla.queue_delay_exceeded`	A queued action exceeded the capability's maximum queue delay	Stop or resubmit as a new authorized operation if appropriate

Code or prefix	Typical meaning	Decision
`storage.unavailable`	Durable runtime storage failed	Retry only within policy; verify durable state before replay
`runtime.handler_failed`	A handler was missing or returned an implementation failure	Diagnose ownership and implementation before relying on retry
`manifest.admission_failed`, `implementation.*`	Manifest or handler ownership admission failed	Fix deployment admission; client retry cannot repair it
`transaction.*`	Plan, commit, compensation, rollback, or reconciliation policy failed	Follow transaction state and its exact recovery branch
`delegation.*`	Target, route, depth, identity, or policy rejected delegation	Correct the delegation contract or authority

Connector-fleet runtime errors

Catalog and central admission

The tenant capability-catalog route uses compact HTTP errors:

Code	HTTP status	Decision
`connector_catalog.disabled`	404	Configure the catalog; client retry cannot enable it
`connector_catalog.tenant_required`	403	Bind the authenticated actor to a verified tenant
`connector_catalog.invalid_query`	400	Correct capability, profile, cursor, text, or limit input
`connector_catalog.stale_cursor`	409	Discard the cursor and restart at the first page
`connector_catalog.unavailable`	503	Retry the read with bounded backoff

Central remote routing and admission return structured protocol errors where the condition has a stable client decision:

Code	Category and retry	Decision
`connector.request_rejected`	`permanent`, false	Correct invalid limits or oversized work
`connector.scheduler_overloaded`	`temporary`, true, 100 ms	Retry only within tenant fairness and action retry policy
`connector.admission_capacity`	`temporary`, true, bounded registry delay	Wait for the exhausted admission scope; do not bypass the quota

Binding absence, replica unavailability, stale registry state, fence loss, and storage failures can surface through a broader runtime error when no dedicated protocol mapping exists. Operators should inspect registry and fleet state rather than infer a specific code from generic handler text.

Connector-host execution

Authenticated host requests receive a signed error envelope for these conditions:

Code	HTTP	Category and retry	Decision
`connector_host.not_ready`	503	`temporary`, true, heartbeat interval	Wait for a valid lease and ready non-draining host
`connector_host.capacity`	429	`temporary`, true, 100 ms	Respect local concurrency; do not scale by bypassing admission
`connector_host.approval_authorization_unavailable`	503	`temporary`, true, 1,000 ms	Recover the approval journal and retry within policy
`connector_host.approval_authorization_conflict`	403	`auth`, false	Reconcile the signed approval with durable host state
`connector_host.credential_revision_unavailable`	503	`temporary`, true, 1,000 ms	Restore credential-revision resolution without changing the pin

Code	HTTP	Category and retry	Decision
`connector_host.credential_revision_denied`	403	`auth`, false	Correct tenant, instance, capability, or revision authorization
`connector_host.route_mismatch`	403	`auth`, false	Discard the request and re-resolve the exact admitted route
`connector_host.authentication`	401	`auth`, false	Correct gateway identity, DID, signature, or trust domain
`connector_host.replay`	409	`policy`, false	Do not reuse the message ID; inspect prior execution state
`connector_host.execution`	502	`connector`, false	Diagnose the host runtime or unsupported message before retry

Schema/decode failures and response-signing failures occur before a trustworthy signed response can be produced. Those paths use compact unsigned JSON with 400 or 500. Treat an unsigned failure as transport evidence only; never accept its body as authenticated host state.

Event and stream-callback ingress

Code	Category and retry	Decision
`connector_event.capacity`	`temporary`, true, 100 ms	Retry the same idempotent event envelope within ingress bounds
`connector_event.invalid`	`permanent`, false	Correct envelope type, size, route, event, or chunk shape
`connector_event.not_authorized`	`auth`, false	Correct recipient, signer, route assignment, tenant, or binding
`connector_event.idempotency_conflict`	`permanent`, false	Stop: the event ID already owns different retained content
`connector_event.unavailable`	`temporary`, true, 1,000 ms	Recover registry or durable event state before bounded retry
`connector_event.not_configured`	`permanent`, false	Configure central ingress before sending callbacks

Registry and admission operator failures

Registry and admission CLI commands fail through standard error and non-zero exit status. Their Rust error families are not `ProtocolError` codes and do not carry a wire-level retry flag.

Failure family	Meaning	Operator action
Admission `Invalid`	Package, policy, identity, limit, or invariant is invalid	Correct the document; do not retry unchanged
Admission `Signature`	Package or evidence signature cannot be verified	Stop and correct the signer, DID, signature, or canonical object
Admission `Evidence`	Required evidence is missing, stale, mismatched, revoked, or rejected	Replace or regenerate the exact evidence set
Admission `Registry`	Applying the verified package failed in catalog storage	Read the durable admission operation before safely resuming
Admission `Journal`	The durable operator journal failed	Restore journal authority and inspect partial state
Admission `Conflict`	Package identity, revision, digest, or terminal revoke conflicts	Reconcile the exact recorded operation; never overwrite it
Registry `BindingUnavailable`	No enabled tenant binding owns the capability	Correct or enable the admitted binding
Registry `ReplicaUnavailable`	No ready replica has capacity	Restore host lease/readiness or scale through admitted orchestration
Registry `CapacityExceeded`	A hard admission counter reached its limit	Respect its bounded retry delay or change approved capacity
Registry `StaleCursor`	Catalog revision changed during pagination	Restart the catalog query from the first page

Failure family	Meaning	Operator action
Registry `FenceLost`	Route assignment fencing failed	Discard the route and resolve a new fenced assignment
Registry `Storage`	Durable registry access failed	Recover storage, then inspect the exact operation before retry

Use `connector registry plan` to verify without writing. `apply` journals and resumes an exact package revision and digest; `status` is the recovery source. Do not repeatedly invoke `apply` based only on an error string.

Callback and delivery failures

Action execution, callback delivery, and event ingestion are separate durable boundaries. A completed action can coexist with a failed callback. Recover the callback from its durable outbox or event state; do not rerun the provider action to compensate for delivery failure.

Likewise, a successful transport response can contain a failed action result, and a failed transport exchange can leave an action running. Read action, transaction, event, and receipt state before choosing another mutation.

Security and observability

- Log code, category, action ID, transaction ID, correlation ID, retry decision,

attempt, and redacted source metadata.

- Do not log credentials, raw provider bodies, bearer tokens, signing seeds, or

unreviewed details.

- Keep one metric label set per bounded code family; do not label by free-form

message, tenant, provider request ID, or resource ID.

- Treat signature, tenant, route, credential-revision, approval-conflict,

replay, and fence failures as security events, not ordinary availability.

- Preserve durable receipts and provider operation references for an unknown

outcome or contested retry.

Related reference

- Native HTTP API
- Connector fleet HTTP API
- Transport bindings
- `getaip` CLI
- Actions and sessions