Hermes streaming cursors, deadlines, and cancellation

Use this reference when a Hermes-backed Action must survive a delivery disconnect, deadline, or cancellation request. It defines the native AIP action-event position and the evidence available after interruption.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/reference/streaming-cursors-and-cancellation.md

The AIP cursor resumes delivery. It does not restart provider execution, guarantee exactly-once consumption, or prove that cancellation stopped a model, tool, message, memory, or network effect.

Position types

These positions are not interchangeable.

Position	Example	Owner	Valid use
AIP action-event cursor	`aev_cursor_42_17`	AIP runtime	Resume the same Action event route
Legacy event-only cursor	`evt_cursor_42`	AIP runtime	Accepted input without a retained chunk position
AIP chunk sequence	`17`	AIP lifecycle store	Order chunks inside one Action
Native SSE event ID	AIP event ID or `chunk:17`	`aipd` renderer	Identify one delivered SSE frame, not route position
Hermes SSE `id`	Provider-defined or absent	Hermes endpoint	Interpret an upstream frame only
Responses `sequence_number`	Provider integer	Hermes Responses API	Order provider Responses events
Structured-run event sequence	Provider integer	Hermes run store	Page provider run events through `run_events`

Persist the cursor emitted by the `aip.stream.cursor` frame. Keep provider IDs and sequences with their payloads, but never submit them as an AIP route cursor.

Native routes

The action-scoped routes are:

```
TEXT
GET /aip/v1/actions/{action_id}/events
GET /aip/v1/actions/{action_id}/chunks
POST /aip/v1/actions/{action_id}/cancel
```

The events route accepts these query parameters:

Parameter	Type	Default	Behavior
`cursor`	String	Start of retained view	Opaque action-event continuation
`limit`	Integer	`100`	Clamped to `1..1000` across events and chunks

Parameter	Type	Default	Behavior
`kind` or `kinds`	String list	All event kinds	Filters stored AIP events, not included chunks
`include_chunks`	Boolean	`false`	Adds chunks after the selected events
`follow`	Boolean	`false`	Returns native SSE and polls until terminal

The chunks route fixes the event-kind filter to `aip.stream.chunk` and enables chunk inclusion. Both routes use the same combined cursor implementation.

Reading requires visibility to the Action and its tenant. Cancellation is a separate authorized lifecycle write.

Cursor grammar

The current runtime emits:

```
TEXT
aev_cursor_<event_offset>_<last_chunk_sequence|none>
```

For example:

```
TEXT
aev_cursor_42_17
aev_cursor_42_none
```

The route also accepts the legacy event-only form:

```
TEXT
evt_cursor_<event_offset>
```

Treat all forms as opaque. The event offset is an implementation position in the retained event store, not the number of matching Action events returned to the caller. Filtering can advance it across records that are not returned.

A legacy cursor carries no chunk position. With `include_chunks=true`, the runtime can return retained chunks from the beginning. Prefer the combined cursor once the route has emitted one.

Malformed prefixes, missing components, non-numeric event offsets, and invalid chunk sequences return `query.invalid_cursor`. An event position outside the retained replay window returns `stream.cursor_expired`.

Page selection

One action-event read proceeds in this order:

1. parse the event and chunk positions;
2. read stored events for the Action using the event position and kind filters;
3. subtract the returned event count from the page limit;
4. when requested, append chunks whose AIP sequence is greater than the chunk

position until the remaining row capacity is used;

5. return the advanced combined cursor and current Action terminal flag.

The default page has at most 100 combined records. The caller's requested limit is clamped to 1,000. Stored event selection also stops before exceeding a 4 MiB event-page bound.

The JSON response has separate arrays:

```
JSON
{
```

```
  "action_id": "act_0123456789abcdef0123456789abcdef",
  "events": [],
  "chunks": [],
  "cursor": "aev_cursor_42_17",
  "terminal": false
}
```

Events are returned before chunks. The arrays are not merged into one chronological sequence. Within `chunks`, order by the AIP `sequence`; within `events`, retain the store order. Use domain identifiers to correlate an event and chunk rather than comparing their array positions.

The kind filter applies only to `events`. Setting `kind=aip.stream.chunk` does not replace `include_chunks=true` when the caller needs lifecycle chunks.

Native SSE frames

With `follow=true`, `aipd` renders repeated action-event pages as SSE.

SSE event name	SSE `id`	Data	Resume meaning
Stored AIP event kind	AIP event ID	`{ "event": ... }`	One record only; not a route cursor
`aip.stream.chunk`	`chunk:<sequence>`	`{ "chunk": ... }`	One chunk only; not a route cursor
`aip.stream.cursor`	Combined AIP cursor	`{ "next_cursor": ... }`	Persist this value for route resume
`aip.stream.terminal`	Combined AIP cursor	Action ID and terminal flag	Delivery reached a terminal Action view
`aip.error`	No resumable ID	Typed protocol error	Stop and apply the error decision

The follow implementation polls every 500 milliseconds when no frame is ready. The outer SSE connection sends a `keep-alive` comment every 15 seconds. Neither value is an Action SLA or a provider heartbeat.

Follow stops after the Action view is terminal or after a protocol error. The terminal frame means the lifecycle is terminal; read status or result for its authoritative outcome and receipts.

`Last-Event-ID` handling

The HTTP binding selects the query `cursor` first. Only when it is absent does it read the `Last-Event-ID` header.

This precedence lets a client override an automatically retained SSE event ID with its last verified combined cursor. That matters because a connection can break after an AIP event ID or `chunk:` frame but before the following `aip.stream.cursor` frame.

Safe persistence rules:

1. accept a cursor only from `aip.stream.cursor.data.next_cursor` or the

terminal frame's combined ID;

2. commit application processing before committing that cursor;
3. reconnect to the same Action route and authorization context;
4. send the committed value as the `cursor` query parameter;
5. tolerate a repeated event or chunk and reject conflicting duplicate data.

Do not persist the latest arbitrary SSE `id`. Do not construct a combined cursor from an AIP event ID and chunk sequence.

Cursor error decisions

Error or observation	Meaning	Decision
`query.invalid_cursor`	The supplied value is not a valid route cursor	Restore the last committed combined cursor; do not guess a position
`stream.cursor_expired`	The event position is outside retained replay	Read Action status/result, record the delivery gap, and reconcile required effects
Empty non-terminal page	No new matching record is currently retained	Continue bounded polling or follow; do not submit a replacement Action
Repeated AIP event ID	Delivery replayed a stored event	De-duplicate by event ID and compare content
Repeated chunk sequence	Delivery replayed a chunk	Accept only identical content for that Action and sequence
Conflicting chunk sequence	Retained evidence is inconsistent	Stop consumption and investigate storage/publication integrity
Terminal frame without expected provider evidence	AIP lifecycle settled before the client saw all expected provider detail	Read the result and preserve the evidence gap

Cursor expiry is a delivery-evidence gap. It is not evidence that the Action did not run.

Deadline boundaries

The connector publishes these timeout classes:

Capability family	Published timeout	Connector request boundary
`chat_stream`, `responses_stream`, `run_events`, `session_chat_stream`	300,000 ms	Trusted deadline or cancellation can stop earlier; default client permits up to 900,000 ms
`operator`	900,000 ms	Operator deadline, policy timeout, and provider state can settle earlier
Non-streaming Hermes operation	60,000 ms	Some operation-specific clients use a shorter or longer request timeout

Published values are contract declarations, not observed latency or availability. A trusted execution deadline is carried separately from the connector client's HTTP timeout. Reaching either boundary does not roll back work already accepted by Hermes or an external tool.

Native Action cancellation

`POST /aip/v1/actions/{action_id}/cancel` accepts an optional JSON reason:

```
JSON
{
  "reason": "The requester no longer needs this operation"
}
```

The runtime performs these lifecycle actions:

1. authorize a write against the target Action;
2. if the Action is currently active, wake its trusted cancellation token and

invoke the active connector cancellation callback;

3. mark the Action lifecycle and queued record cancelled;
4. append `aip.action.cancelled`;
5. return an AIP `ActionResult` with status `cancelled` and

`{ "cancelled": true }` output.

The connector callback is active-only. Cancelling an inactive record can change the AIP lifecycle without calling Hermes. The returned result is an AIP acknowledgement, not provider terminal evidence.

Closing a follow connection performs none of these steps.

Hermes cancellation effects

Active Action family	Connector/provider effect	Required reconciliation
Chat or Responses stream	Wakes the connector token and drops the active provider stream; Hermes attempts to interrupt its task after disconnect	Read original AIP result; inspect session, stored response, tools, and delivery effects
Session chat stream	Drops the active session stream cooperatively	Read the transcript and external effects; do not infer rollback
Structured `run_events` read	Stops the observation Action, not necessarily the bound Hermes run	Read `run_status`; invoke separately governed `run_stop` when provider interruption is required
Active operator source or resume	Records cancellation intent, maps resume to source, attempts provider stop, and reconciles within its grace	Inspect operator binding and provider state; native cancel output omits the provider classification
Operator already at `requires_human`	No active connector callback exists	Use a governed `deny` resume for the prompt or separately approved `run_stop` for full interruption

Provider completion can win a race with cancellation. Conversely, the AIP lifecycle can report cancelled while provider state remains unknown. Preserve both observations and do not replace one with the other.

`run_stop` is a provider mutation with its own capability contract, approval, Action ID, and idempotency key. It is not an alias for native Action cancel.

Security and evidence

Action-event reads and cancellation use the transport-authenticated principal and Action ownership or tenant context. Do not move a cursor to another tenant, principal, Action, route, or deployment.

Event and chunk payloads can contain prompts, model output, reasoning, tool arguments, results, personal data, and provider identifiers. Encrypt retained cursors with their application record, but redact or minimize the payloads stored beside them.

For an interrupted operation, retain:

- deployment and endpoint identity;
- capability and Action ID;
- last committed combined cursor;
- last accepted AIP chunk sequence and event IDs;
- cancellation reason and timestamp when present;
- final AIP status/result and receipts;
- separately verified Hermes and external-system state.

This evidence is source-aligned. It does not establish live cancellation, durable retention, provider availability, or production qualification.

Related documentation

- Stream chat and Responses
- Run an operator lifecycle
- Structured runs and approvals

- Use native AIP
- Transport bindings
- Errors and recovery