Stream Hermes chat and Responses

Use this guide to submit one streaming Hermes action, render incremental output, and reach a verified AIP result. It covers chatstream and responsesstream on an endpoint already admitted to the tenant catalogue.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/guides/chat-and-response-streaming.md

Streaming can execute models, tools, memory writes, and message delivery. Partial text, a terminal-looking provider frame, and a final AIP result are different evidence. Never replay a stream merely because one delivery connection closed.

Understand the four views

One model turn appears in four nested views.

View	Owner	What it proves
Hermes SSE frame	Hermes endpoint	One upstream role, text, tool, finish, or terminal signal
AIP `StreamChunk`	AIP runtime	One parsed provider frame was recorded for this action and sequence
Native follow SSE frame	`aipd`	A stored event or chunk was delivered to this consumer
Final `ActionResult`	AIP runtime	The connector classified the collected stream as completed, failed, or cancelled

The outer follow stream can disconnect while the action and Hermes request continue. Conversely, an upstream Hermes stream can end while the AIP follow connection remains open long enough to deliver the final failure.

Use the AIP action ID as the recovery identity. Use provider IDs inside events to interpret Chat, Responses, tool, and continuity state. Do not substitute one for the other.

1. Choose Chat or Responses

Need	Capability	Terminal provider signal
Stream an OpenAI-compatible message turn	`cap:hermes_agent:<endpoint_id>:chat_stream`	`data: [DONE]` after a finish chunk
Stream a stored or chainable Responses turn	`cap:hermes_agent:<endpoint_id>:responses_stream`	`response.completed` or `response.failed`

Choose Responses with `store: true` when an application needs a provider record that can be reconciled after cancellation or upstream disconnect. Choose Chat when its message shape and explicit Hermes session continuity are the intended interface.

Both capabilities are stable, medium risk, retry-unsafe, and not subject to a connector retry. Both advertise synchronous, asynchronous, streaming, and cancellation support, but `async_expected` is false. The provider

does not continue as a background delivery service after its HTTP stream is gone.

The published capability timeout is 300 seconds. The connector's default HTTP client permits up to 900 seconds, but a trusted AIP deadline or explicit cancellation can stop the request earlier. Use the final AIP state as the outcome boundary; do not infer success from elapsed time.

2. Allocate stable AIP identity

Generate and retain the AIP action ID before submission. If the application uses the optional idempotency field, allocate one key for this exact request and retain it with the action ID.

An optional key does not make model or tool execution safe to retry. Provider streaming bypasses the in-memory idempotency cache used by non-streaming Chat and Responses calls.

Record at least:

- the endpoint-qualified capability ID;
- the AIP action ID and optional idempotency key;
- the tenant and authenticated principal used for submission;
- the chosen continuity fields;
- the durable follow cursor returned by `aipd`.

Do not generate a replacement action ID after an uncertain submission. Query the original ID first.

3. Prepare a bounded Chat stream

Use one Chat input branch and an explicit Hermes session ID:

```
JSON
{
  "messages": [
    {
      "role": "system",
      "content": "Return a concise operational summary."
    },
    {
      "role": "user",
      "content": "Summarize the current incident notes."
    }
  ],
  "model": "hermes-agent",
  "session_id": "incident-review-2030-01",
  "session_key": "support:incident-review",
  "max_events": 512
}
```

The connector forces `stream` to `true`. It removes continuity and trusted transport fields from provider JSON and sends them as headers.

Use an explicit `session_id` from the first turn. When it is absent, Hermes derives one and returns it only in an HTTP header. The current connector keeps provider SSE data but does not expose those response headers in its aggregate.

`max_events` limits parsed provider frames collected by the connector. It does not limit output tokens, tool calls, AIP follow pages, or model execution. The default is 4,096 frames. A low value is a safety cutoff, not normal completion.

4. Prepare a recoverable Responses stream

Keep the provider body separate from connector controls:

```
JSON
{
```

```
  "body": {
    "model": "hermes-agent",
    "input": "Summarize the current incident notes.",
    "instructions": "Return three concise findings.",
    "store": true,
    "truncation": "auto"
  },
  "session_key": "support:incident-review",
  "max_events": 512
}
```

The connector forces the provider body's `stream` field to `true`. Put `max_events`, `session_key`, and the AIP idempotency value outside `body`.

With `store: true`, Hermes persists an `in_progress` snapshot immediately after `response.created`. It later replaces that snapshot with completed, failed, or incomplete provider state. Capture the response ID from the first event so `response_get` can reconcile the provider record.

For chaining, set `previous_response_id`, `conversation`, or explicit `conversation_history` according to the capability reference. Do not combine `conversation` and `previous_response_id`.

5. Submit and follow the same action

Place the chosen input in `request.json`, then start the stable action:

```
SH
export AIP_CAPABILITY_ID='cap:hermes_agent:replace_endpoint:chat_stream'
export AIP_ACTION_ID='act_replace_with_stable_id_001'

cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$AIP_CAPABILITY_ID" \
  --action-id "$AIP_ACTION_ID" \
  --input @request.json
```

In a separate consumer, follow durable events and chunks for that same ID:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action events "$AIP_URL" \
  "$AIP_ACTION_ID" \
  --include-chunks \
  --follow
```

An application can use the equivalent bearer-authenticated route:

```
TEXT
GET /aip/v1/actions/{action_id}/events?include_chunks=true&follow=true
```

The follow loop polls the durable view every 500 milliseconds and emits an outer keepalive every 15 seconds. Hermes independently writes upstream keepalive comments after 30 seconds of inactivity. The connector ignores those comments; they do not consume `max_events` or create AIP chunks.

6. Decode each durable chunk

An outer `aip.stream.chunk` follow event contains an AIP chunk. Its structured payload has this shape:

```
JSON
{
  "action_id": "act_replace_with_stable_id_001",
  "sequence": 1,
  "kind": "data",
  "data": {
    "hermes_sse": {
      "event": null,
      "id": null,
      "data": {
```

```
        "choices": [
          {
            "delta": {
              "content": "The incident"
            },
            "finish_reason": null
          }
        ]
      },
      "raw_data": "{\"choices\":[{\"delta\":{\"content\":\"Hi\"}}]}",
      "terminal": false
    }
  }
}
```

The connector assigns `StreamChunk.sequence` from zero in provider-frame order. The embedded Responses `sequence_number`, when present, is a different provider sequence. Never compare or persist one as the other.

The runtime persists a chunk before publication returns. An identical repeat of the same AIP sequence is accepted as replay. Conflicting content, non-increasing new sequences, and chunks after an AIP terminal chunk are rejected.

Use `data.hermes_sse.data` as the provider event. Keep `raw_data` for controlled diagnostics only; it can contain prompts, output, tool arguments, tool results, and personal data.

7. Render Chat events

Handle Chat frames in this order.

Provider frame	Application action
Initial `choices[0].delta.role`	Open an assistant message; render no text
`choices[0].delta.content`	Append content to that message
`event: hermes.tool.progress`	Update the correlated tool status from its payload
Finish chunk	Save `finish_reason`, usage, and any `hermes` or `error` object
`data: [DONE]`	Mark the provider stream explicit-terminal, then wait for AIP result

Chat content deltas contribute to the final aggregate `text`, but the current connector does not copy them into `StreamChunk.part`. A renderer that reads only `part` will display no Chat text. Read the embedded `choices` delta.

`hermes.tool.progress` carries tool name, correlation ID, and running or completed state. Its provider event name does not match the connector's generic `tool.started`, `tool.completed`, or `tool.failed` matcher. The AIP chunk is therefore normally `kind: data`, not `kind: tool`.

Never discard `kind: data` chunks when tools matter. De-duplicate tool updates by their provider correlation ID and AIP sequence.

8. Validate the Chat finish

The final Chat data chunk can carry one of three `finish_reason` values.

Finish reason	Meaning	Application outcome
`stop`	Hermes reported ordinary completion	Candidate success; still require final AIP completion
`length`	Usable output was truncated	Partial result; do not present it as complete
`error`	Agent execution failed	Failed business result even if some text exists

The provider writes `[DONE]` after all three. The connector treats `[DONE]` as an explicit terminal frame, but its final outcome classifier does not inspect Chat `finish_reason`. An `ActionResult` can therefore have status `completed` while the retained finish chunk says `length` or `error`.

Accept a Chat stream only when all of these are true:

1. the final AIP result is `completed`;
2. aggregate `terminal` is true and `truncated` is false;
3. a retained finish chunk has `finish_reason: stop`;
4. no retained provider error contradicts that finish;
5. any required tool effects have application-level confirmation.

Treat missing finish evidence as an incomplete outcome. Do not infer success from accumulated `text`.

9. Render Responses events

Responses uses named provider events with its own `sequence_number`.

Event	Application action
`response.created`	Capture `response.id` and initial `in_progress` state
`response.output_item.added`	Open a message, function call, or function-call output item
`response.output_text.delta`	Append `delta` to the named message/content index
`response.output_text.done`	Verify the full text for that content item
`response.output_item.done`	Close the identified item and preserve its final payload
`response.completed`	Read the full terminal response and usage
`response.failed`	Read redacted failure data and settle as failed

Responses text deltas appear in both embedded `data.delta` and, when non-empty, `StreamChunk.part`. Prefer the structured provider identifiers so multiple output items cannot be merged accidentally.

Function calls and their outputs use `response.output_item.*` names. These events are retained as AIP `data` chunks rather than generic `tool` chunks. Pair them with `call_id`, `item.id`, and `output_index` from the provider payload.

The connector recognizes `response.completed` as Done and `response.failed` as Error. It also understands an upstream `response.incomplete` name, although the pinned Responses writer normally persists incomplete state on disconnect rather than delivering that terminal event over the broken stream.

10. Read the final AIP result

The connector returns this aggregate inside `ActionResult.output`:

```
JSON
{
  "endpoint_id": "replace_endpoint",
  "operation": "responses_stream",
  "http_status": 200,
  "events": [],
  "text": "Three findings are ready for review.",
  "terminal": true,
  "truncated": false
}
```

The actual `events` array contains every parsed provider frame, including its raw data. `text` is best-effort convenience output. It is not a substitute for the terminal provider response, finish reason, tool state, or final AIP status.

Aggregate state	AIP result	Decision
`truncated: true`	Failed with `connector.hermes_agent.stream_truncated`	Preserve partial evidence; do not replay automatically

Aggregate state	AIP result	Decision
`terminal: false`	Failed with `connector.hermes_agent.stream_incomplete`	Reconcile original action and provider state
Terminal `response.failed`, `response.incomplete`, or `error`	Failed with `connector.hermes_agent.stream_failed`	Inspect typed error and retained provider event
Terminal `run.cancelled`	Cancelled	Confirm external effects separately
Other explicit terminal	Completed	Apply Chat or Responses business checks before accepting

For Chat, apply the additional `finish_reason` rule from the preceding step. For Responses, require the terminal response object's status to agree with the AIP result.

11. Choose the event limit safely

Omitting `max_events` uses the connector default of 4,096 parsed frames. The final aggregate retains all collected events, so choose a lower application bound only when it still leaves substantial headroom for text and tool frames.

The connector publishes and stores the frame that reaches the limit, then sets `truncated` true and `terminal` false. It checks the limit before accepting terminal completion. A terminal provider frame that lands exactly at the configured limit can therefore be published as Done while the final AIP result is Failed for truncation.

The final `ActionResult` wins over a conflicting last chunk. Do not retry with a larger limit after model or tool work may have occurred. Use the original action evidence to reconcile the effect.

An upstream close without an explicit terminal produces a Failed result. With runtime publication active, the connector also appends an AIP Error chunk with reason `upstream_stream_closed_without_terminal_event`. The truncation branch does not append that extra terminal Error chunk, so always read the result.

12. Recover a follow disconnect

Closing or losing the native `aipd` follow SSE does not cancel the action or the upstream Hermes request.

On reconnect:

1. keep the original action ID;
2. resume the same action route with its last retained route cursor;
3. tolerate a repeated outer event or chunk;
4. order chunks by AIP `sequence` and reject conflicting content;
5. read the final action status or result after the follow stream terminates.

Persist only a cursor returned by the same route and authorization context. The exact cursor formats, paging bounds, and expiry decisions belong to the streaming cursor and cancellation reference later in this documentation set.

Do not resume a provider SSE URL directly. Chat and Responses frames do not provide a usable replay cursor in this connector path.

13. Cancel the original action

Closing the renderer is not cancellation. Request cancellation explicitly:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_TOKEN_FILE" \
  action cancel "$AIP_URL" \
  "$AIP_ACTION_ID" \
```

```
  --reason 'The requester no longer needs this stream'
```

The runtime records cancellation and wakes the action's trusted cancellation token. The connector drops the active HTTP stream. When the Hermes writer observes the disconnect, it attempts `agent.interrupt()` and cancels its task.

This is cooperative stop, not effect rollback. A model request, tool call, message, memory write, or stored response may already exist. Read the original AIP result and provider state before deciding that another action is safe.

For `responses_stream` with `store: true`, cancellation can leave a retrievable `incomplete` response snapshot. For Chat, preserve explicit session identity and inspect the session according to the session capability reference; do not assume partial transcript persistence.

14. Reconcile a stored Responses stream

As soon as `response.created` arrives, persist its `response.id`. After an uncertain terminal outcome, call `response_get` once with that ID:

```
JSON
{
  "response_id": "resp_0123456789abcdef0123456789ab"
}
```

Interpret the stored status:

- `completed` supplies a terminal provider object for review;
- `failed` supplies retained failure state;
- `incomplete` preserves partial output after disconnect or cancellation;
- `404` means the response was not stored, was evicted or deleted, or the

request reached another response store.

Do not submit a replacement stream merely because `response_get` returns incomplete. Tool or delivery effects can precede that snapshot.

15. Handle failures

Symptom	Meaning	Decision
No AIP chunks yet	Action may be queued, rejected, or before the first provider frame	Read action status; do not resubmit
Follow SSE disconnected	Delivery failed; action state is unknown to this consumer	Resume from durable cursor and read result
Provider HTTP `429`	Hermes concurrent agent-run gate is full	Preserve original action; follow its typed result
`stream_truncated`	Connector frame cutoff was reached	Keep partial evidence; do not retry model work
`stream_incomplete`	Upstream closed without explicit terminal	Reconcile action and stored Responses state
`stream_failed`	Provider delivered a named terminal failure	Inspect retained terminal event and typed error
AIP result Completed but Chat finish is `length` or `error`	Connector terminal and Chat business finish disagree	Treat as partial or failed, not success
Cancellation returned Cancelled	Runtime stopped waiting and attempted upstream interruption	Verify provider and external effects
`response_get` returns `404`	No retained object is reachable at this store	Check original event ID, `store`, instance, and result

Match protocol error codes, not free-form provider messages. Streaming calls are retry-unsafe even when the visible text is empty.

16. Verify the client

Before accepting the integration, use controlled evidence to verify that the client:

1. retains the action ID before submission;
2. follows the same ID with chunks included;
3. orders and de-duplicates by AIP chunk sequence;
4. renders Chat from `choices` rather than relying on `part`;
5. handles tool events carried as `kind: data`;
6. preserves Responses item and call identifiers;
7. saves `response.id` from `response.created` when storage is enabled;
8. requires explicit provider terminal evidence and final AIP state;
9. rejects Chat `length` and `error` finishes as complete success;
10. handles a terminal frame exactly on `max_events` as truncation;
11. resumes follow delivery without creating a new action;
12. sends explicit cancellation and then reconciles effects;
13. redacts raw frames, prompts, tools, outputs, IDs, and error data.

This is a source-aligned client checklist. It is not live model, latency, availability, cancellation, or production qualification evidence.

Related documentation

- Chat and Responses capabilities
- Hermes capability index
- Persistent sessions
- Use native AIP
- Transport bindings
- Errors and retry decisions