Hermes persistent session capabilities

Use these nine endpoint-qualified capabilities when an application needs an addressable Hermes transcript that can be listed, inspected, renamed, branched, continued, or deleted.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/capabilities/sessions.md

The resource is a Hermes `SessionDB` row and its messages. It is not an AIP Session, a stored Responses object, or a structured Hermes run. The pinned Hermes endpoint stores these resources in its shared SQLite `state.db`.

Session chat can run tools and create external effects. A timeout or broken connection is not evidence that nothing happened, and it is not safe to replay the turn blindly.

Operations

Replace `` with the normalized endpoint ID present in AIP discovery.

Operation	Provider request	Kind	Risk	Approval	Key	Retry
`sessions_list`	`GET /api/sessions`	Tool	Low	No	Optional	Safe
`session_create`	`POST /api/sessions`	Tool	Medium	No	Required	Unsafe
`session_get`	`GET /api/sessions/{session_id}`	Tool	Low	No	Optional	Safe
`session_patch`	`PATCH /api/sessions/{session_id}`	Tool	Medium	No	Required	Unsafe
`session_delete`	`DELETE /api/sessions/{session_id}`	Tool	High	Yes	Required	Unsafe
`session_messages`	`GET /api/sessions/{session_id}/messages`	Tool	Low	No	Optional	Safe
`session_fork`	`POST /api/sessions/{session_id}/fork`	Tool	Medium	No	Required	Unsafe
`session_chat`	`POST /api/sessions/{session_id}/chat`	Agent	Medium	No	Optional	Unsafe
`session_chat_stream`	`POST /api/sessions/{session_id}/chat/stream`	Agent	Medium	No	Optional	Unsafe

All nine operations are stable and require the selected endpoint's Bearer credential. The endpoint comes from the capability ID. Input cannot select a different endpoint or provider URL.

The connector percent-encodes every dynamic `session_id` path segment and rejects carriage return, line feed, and NUL characters before making a request.

Choose the correct continuity model

Need	Use	Durable identity
Address and inspect a Hermes transcript	A capability on this page	Hermes `session_id`
Continue ordinary Chat or a stored Responses chain	Chat and Responses capabilities	Chat `session_id`, Responses ID, or conversation name
Poll work or resume a provider approval	Structured-run capabilities	Hermes `run_id`
Carry protocol-level interaction context	AIP Session	AIP `SessionId`
Scope long-term memory across transcripts	`session_key` on a chat call	`X-Hermes-Session-Key` value

These identifiers are independent. A path `session_id` selects one persisted transcript. A `session_key` scopes optional long-term memory and does not select, create, rename, or authorize that transcript.

An AIP Action ID and idempotency key govern one AIP invocation. Neither is a Hermes transcript ID. See Actions and Sessions for the protocol-level distinction.

Shared AIP contract

Contract field	Read operations	Create, patch, and fork	Delete	Chat operations
Side effects	`read`, `external_network`	`read`, `write`, `external_network`	`read`, `delete`, `external_network`	`read`, `write`, `send_message`, `external_network`
Published data class	Internal; no PII	Confidential; contains PII	Restricted; contains PII	Confidential; contains PII
Redaction	Required	Required	Required	Required
Idempotency	Optional	Required	Required	Optional
Retry support	Yes	No	No	No
Retry safety	Safe	Unsafe	Unsafe	Unsafe
Expected completion	Sync	Sync	Sync	Sync or streaming
Transaction	None	None	None	None
Compensation	Rollback not supported	Rollback not supported	Rollback not supported; approval required	Rollback not supported

Every operation supports synchronous AIP execution. `session_chat_stream` also advertises streaming and cancellation, with streaming expected completion. It does not advertise async support, and `async_expected` is false.

The contract declares a 60-second timeout for non-streaming operations and a 300-second timeout for the stream. The connector's HTTP timeout is 300 seconds for `session_chat` and 30 seconds for every other JSON operation. The streaming request does not override the default connector client's 900-second timeout. Trusted execution is also bounded by its AIP deadline and cancellation token.

Expected latency is 5 seconds for non-streaming operations and 30 seconds for the stream. The maximum queue-delay hint is 10 seconds, and the availability target is `99.9%`. These are contract declarations, not live performance or qualification evidence.

List sessions

Use `sessions_list` to enumerate eligible rows from the endpoint's shared session database:

```
JSON
{
  "limit": 50,
  "offset": 0,
  "source": "api_server",
  "include_children": false
```

```
}
```

`limit` defaults to 50 and is capped at 200. `offset` defaults to zero and is capped at 1,000,000. The provider orders results by most recent activity.

By default, compression continuations and delegate children are hidden. Branch sessions remain listable. Set `include_children` to `true` only for an administrative view that can distinguish those row types.

Archived rows are also excluded. The pinned session-list route has no query parameter that includes or selects archived rows.

The optional `source` filter is exact. A session created through this API uses `api_server`, but the same database can contain CLI, gateway, cron, and other Hermes sessions. An unfiltered list is not an inventory of AIP-created rows.

The connector also accepts a `query` object and forwards its scalar keys. Use the typed top-level fields where possible. Supplying the same field both inside `query` and at the top level produces duplicate query parameters.

A result has this provider shape:

```
JSON
{
  "object": "list",
  "data": [
    {
      "id": "support-case-2030",
      "source": "api_server",
      "model": "hermes-agent",
      "title": "Support case 2030",
      "message_count": 4,
      "last_active": 1893456120.0,
      "preview": "Review the current case notes",
      "has_system_prompt": true,
      "has_model_config": false
    }
  ],
  "limit": 50,
  "offset": 0,
  "has_more": false
}
```

`has_more` is `true` whenever the returned page length equals `limit`. It is not an exact total-count calculation and can be true on the final full page. Avoid a zero limit, advance `offset` by the returned count, and stop on a shorter or empty page.

Pagination is offset-based over a changing last-activity order, not a snapshot or cursor. Concurrent activity can move a row between pages. De-duplicate by session ID, retain `_lineage_root_id` when present, and re-list when a complete administrative inventory matters.

When a root was ended by context compression, the list can project it to its latest continuation. The visible `id`, counts, title, preview, and activity then belong to the tip. `_lineage_root_id` retains the original root identity, while `started_at` remains the root's original start time.

Create a session

`session_create` accepts a provider-native object. A top-level `body` wrapper keeps transport fields separate:

```
JSON
{
  "body": {
    "id": "support-case-2030",
    "title": "Support case 2030",
    "model": "hermes-agent",
    "system_prompt": "Keep responses factual and concise."
  }
}
```

Put the required AIP idempotency key on the Action, not inside `body`. The connector unwraps `body`, sends the endpoint credential, and forwards the key as `Idempotency-Key`.

The provider selects the first non-empty `id` or `session_id`. If neither is truthy, including when an empty string is supplied, it generates `api__`. A selected ID must remain non-empty after trimming, be no longer than 256 characters, contain no control characters, and contain no `..`, slash, backslash, or leading Windows drive prefix.

Titles are trimmed, control characters are removed, whitespace is collapsed, and empty text becomes no title. A title must be unique across the shared database and cannot exceed 100 characters. Invalid titles make creation fail; the handler removes the row it just created before returning the error.

The optional model defaults to the endpoint's advertised model. A `system_prompt` must be a string. Unknown provider fields are not rejected by the handler, but they do not become durable fields merely because the open AIP schema accepts them.

Create stores `model` and `system_prompt` as session metadata. The pinned session-chat handler does not use the create request's model as a per-turn model route, and a stored system prompt is not a substitute for the current turn's `system_message`. Model selection and prompt restoration still follow Hermes runtime policy.

The response exposes a client-safe projection. It reports `has_system_prompt` and `has_model_config` rather than returning their stored contents.

Get or patch metadata

Read one exact row with:

```
JSON
{
  "session_id": "support-case-2030"
}
```

`session_get` does not follow a compression continuation. It returns the exact row selected by the path. Use the list projection or the `session_id` returned by `session_messages` to discover a current tip.

Patch only `title` and `end_reason`:

```
JSON
{
  "session_id": "support-case-2030",
  "body": {
    "title": "Support case 2030 — reviewed",
    "end_reason": "client_closed"
  }
}
```

A null or empty title clears it. An empty `end_reason` is ignored. A non-empty reason sets `ended_at` and `end_reason` only when the session is still open. The first close reason wins; patch does not reopen a closed session.

An empty patch body is a successful no-op and returns the current safe session projection.

Closure is metadata, not an authorization state. The pinned chat handlers check that the row exists but do not reject an ended session or reopen it. An application that treats `end_reason` as final must prevent further chat itself.

The provider rejects every other patch field with `unsupported_session_field`. It does not expose an API patch for the stored model, system prompt, token counts, parent, or message history.

Read active messages

Call `session_messages` with the selected `session_id`. Unlike `session_get`, this operation resolves a compression chain before reading. Always inspect the top-level `session_id` in its result:

```
JSON
{
```

```
  "object": "list",
  "session_id": "support-case-2030-continuation",
  "data": [
    {
      "id": 41,
      "session_id": "support-case-2030-continuation",
      "role": "user",
      "content": "Summarize the verified findings.",
      "timestamp": 1893456100.0
    },
    {
      "id": 42,
      "session_id": "support-case-2030-continuation",
      "role": "assistant",
      "content": "Three findings are verified.",
      "finish_reason": "stop",
      "timestamp": 1893456120.0
    }
  ]
}
```

Messages are returned in database insertion order. The route returns active rows only and has no pagination parameter. It can expose content, tool calls, tool results, reasoning, token counts, and finish metadata, but it omits private database columns outside the client-safe allowlist.

After resolution, the route reads only the selected descendant row. It does not concatenate messages from compression ancestors into one API response.

Soft-deleted or compaction-archived rows are not returned. The API does not provide a switch to include them.

Fork a transcript

Use `session_fork` to create a child containing a copy of the source's active messages:

```
JSON
{
  "session_id": "support-case-2030",
  "body": {
    "id": "support-case-2030-alternative",
    "title": "Support case 2030 — alternative"
  }
}
```

The provider marks the source as ended with `branched` when it is still open. It creates an `api_server` child, copies the source model and system prompt, sets `parent_session_id`, and copies active messages. It does not copy inactive history or the complete source row.

Fork does not require an open source. When the source already has another close reason, the first reason remains. The child is still created, but the default child-filtered list may not recognize it as a visible branch. Retain its ID and read it directly, or inspect an administrative list with children included.

If no child ID is supplied, the provider generates one. If no title is supplied, it derives the next unique lineage title, such as `Case #2`.

Fork is not atomic in the pinned provider. The source is ended, the child is created, and messages are copied before the title is validated. A duplicate or invalid title can therefore return HTTP 400 after durable changes occurred. After any fork error, list and inspect both source and intended child before deciding whether another Action is safe.

An absent or empty fork ID triggers provider generation. A selected whitespace-only ID, or one containing CR, LF, or NUL, is rejected. The fork handler does not apply the create handler's full path-safety or 256-character checks. Applications should apply the stricter create-ID rules to both operations.

Continue a session with chat

`session_chat` runs one turn against the stored active history:

```
JSON
{
  "session_id": "support-case-2030",
  "message": "Summarize only the findings supported by the transcript.",
  "system_message": "Return at most three bullets.",
  "session_key": "support:case-review"
}
```

The provider requires a visible `message` or `input`. The connector also maps a non-empty top-level `prompt` or `query` to `message`. A non-empty `system_message` takes precedence over `instructions` and applies as an ephemeral system instruction for this turn.

Text can be a string or a list of text and image parts. Hermes accepts `text`/`input_text` and `image_url`/`input_image`, with HTTP, HTTPS, or `data:image/...` image URLs. File parts and other content types fail with HTTP 400.

A string and each individual text part are capped at 65,536 characters, and the content list is capped at 1,000 parts. The pinned multimodal normalizer does not impose a second aggregate text cap across all accepted parts. The 10,000,000-byte request limit remains the outer bound.

`session_key` becomes `X-Hermes-Session-Key`. Hermes accepts that header only when its API key is configured, because the key selects a long-term-memory scope. It does not replace path-based transcript identity.

The agent receives the exact path session ID and attempts to load its active conversation. If that history read raises, the handler logs the failure and continues with empty history.

The agent is wired to the shared `SessionDB` and attempts to persist the new turn. Session creation and message-append failures are logged and do not necessarily fail the model response. A successful chat result therefore does not prove that old history was loaded or that the new turn is durable. Verify with `session_messages`.

Context compression can rotate the effective transcript during the turn. Use the returned `session_id` for the next call.

A successful provider result has this shape:

```
JSON
{
  "object": "hermes.session.chat.completion",
  "session_id": "support-case-2030",
  "message": {
    "role": "assistant",
    "content": "Three findings are supported by the transcript."
  },
  "usage": {
    "input_tokens": 240,
    "output_tokens": 12,
    "total_tokens": 252
  }
}
```

The connector places that object in the completed AIP ActionResult output. A provider response header also carries the effective session ID, but the JSON field is the portable value available in connector output.

Model execution may call tools, send messages, or mutate external systems. Neither closing the HTTP connection nor deleting the transcript compensates those effects.

Bound streaming behavior

`session_chat_stream` accepts the same turn fields and returns the connector's stream aggregate. On a successful turn, the provider emits `run.started`, `message.started`, assistant deltas, tool progress, `assistant.completed`, `run.completed`, then `done`. Each data event carries a generated run ID, sequence, session ID, and timestamp.

The provider writes an SSE keepalive comment after 30 seconds without an event. The connector retains at most 4,096 parsed events by default. A positive `max_events` can lower that bound; reaching it returns `truncated: true` and `terminal: false`.

Do not interpret truncation, disconnect, or cancellation as rollback. The streaming guide and cursor reference own detailed publication, terminal, and cancellation decisions.

Understand persistence and lineage

The provider lazily opens `HERMES_HOME/state.db`. It uses the same SQLite store as other local Hermes interfaces, with WAL and application-level write retries. There is no separate AIP session database inside this connector.

If the database cannot be opened, resource and session-chat routes return `session_db_unavailable` with HTTP 503 before starting their task. The pinned session API does not fall back to an in-memory transcript store.

Deleting a session removes its row and messages. Delegate descendants are cascade-deleted. Branch and compression children are preserved and have their `parent_session_id` cleared.

The API handler does not pass a transcript-directory path into `SessionDB` deletion. Separate `.json`, `.jsonl`, and `request_dump_*` files are therefore not removed by this route. Deletion also does not remove long-term memory, stored Responses objects, external tool effects, or provider data outside this database.

SQLite persistence is local to the configured Hermes home. The pinned code does not replicate session rows across endpoint instances. Route a given session to the endpoint that owns its database, or provide a deployment-level shared-storage design and validate its concurrency behavior separately.

Protect transcript data

The AIP manifest classifies `sessions_list`, `session_get`, and `session_messages` as Internal and `contains_pii: false` because they are marked read-only. That mechanical classification is weaker than the actual outputs.

Lists can contain `user_id` and a message preview. Session rows expose activity and cost metadata. Message reads can contain user text, assistant text, reasoning, tool arguments, and tool results. Treat all three outputs as confidential and potentially PII-bearing, and redact them from logs and traces.

The safe provider projection hides full stored system prompts and model configuration, but it is not a general content-redaction layer. Apply tenant authorization before exposing any session selected by caller-controlled ID.

The Bearer credential authenticates access to the Hermes API, not ownership of an individual session row. The pinned handlers do not compare the AIP principal or tenant with a row's `user_id` before list, read, patch, fork, chat, or delete. Isolate the endpoint and `state.db` at the intended tenant boundary, or add a separate row-authorization layer before public exposure.

Apply idempotency and retry rules

`session_create`, `session_patch`, `session_fork`, and `session_delete` require an AIP idempotency key. The runtime scopes it to the capability, retains it for 24 hours, and uses `ReturnOriginalResult` on collision.

Within that window, reuse of the same scoped key returns the stored result even when the new Action has different input. This collision mode does not revalidate the retained input hash. Use a fresh key for every changed session, body, or business intent.

`session_delete` also requires tenant-policy approval. That approval lasts 15 minutes and requires a reason, input snapshot, and policy-decision evidence.

Hermes does not enforce `Idempotency-Key` on these session handlers. The AIP runtime is the replay boundary. Bypassing AIP or losing its idempotency store removes that protection.

Only list, get, and messages are published as retry-supported and safe. For a transport failure or remote 5xx on a mutation, reconcile provider state before submitting a new Action. For chat, inspect messages and downstream systems, but do not assume that absence of a final assistant row proves that no tool ran.

The generic connector marks non-read transport failures, deadlines, remote 429, and remote 5xx as uncertain outcomes. The fork title failure described above is a stricter exception: the provider returns 400 after possible writes, while generic connector metadata classifies that status as a definitive remote rejection. Reconcile it manually.

That non-retryable 400 result can also be settled under the fork's idempotency key. Reusing the key then returns the stored failure; it does not inspect or repair the partially created child.

See Approvals and Policy for the delete approval record and Errors and Retry Decisions for the shared AIP failure model.

Handle provider failures

Signal	Meaning	Application decision
`400 invalid_session_id`	A selected create or fork ID is unsafe, too long where checked, or whitespace-only	Correct the ID; inspect state first after fork
`400 invalid_title`	Title is invalid or already used	Choose a valid unique title; inspect state first after fork
`400 unsupported_session_field`	Patch contains a non-public field	Send only `title` and `end_reason`
`400 missing_message` or content error	Session chat has no usable text/image or uses an unsupported part	Correct the current turn; do not alter prior history
`401 invalid_api_key`	Endpoint credential is absent or wrong	Repair endpoint authentication; do not retry unchanged
`403` for `X-Hermes-Session-Key`	Hermes has no API key configured for memory scoping	Configure authenticated access or omit the memory key
`404 session_not_found`	The exact path row does not exist	Re-list, resolve any compression tip, and verify endpoint routing
`409 session_exists`	Create or fork child ID already exists	Read the existing row before choosing a new identity
`413 body_too_large`	JSON body exceeds 10,000,000 bytes	Reduce current input; do not split one mutation into blind retries
`503 session_db_unavailable`	Hermes could not open its shared session database	Restore storage, then reconcile before mutation retry
Connector timeout or transport failure	Completion is unknown for non-read work	Inspect session and external effects before any new Action

The pinned session-chat handlers increment the in-flight run counter but do not call the provider's shared concurrency-limit check used by ordinary Chat, Responses, and structured runs. Do not infer that the published queue-delay or availability hints enforce a session-specific concurrency ceiling.

Verify an integration

Before relying on this family in an application:

1. Discover all nine capability IDs for the intended endpoint and verify the pinned connector revision.
2. Confirm the endpoint credential and tenant routing described in Authentication and Endpoints.
3. Create a uniquely named session with an AIP idempotency key, then get it and locate it in a filtered list.
4. Patch its title, read the result, and verify that unknown fields are rejected.
5. Submit one harmless chat turn, verify the returned effective session ID, and read the active message sequence.
6. Fork the session with a unique child ID and title. Verify the parent end state, child parent ID, and copied active transcript.
7. Exercise delete only with tenant-policy approval and a fresh idempotency key. Verify preserved branches and any retained external transcript artifacts.
8. Record Action IDs, idempotency keys, endpoint identity, status, and redacted evidence without logging transcript content or credentials.

For general model requests, compare Chat and Responses. Return to the Hermes capability index when another operation family owns the task.