Manage a persisted Hermes session lifecycle

Use this guide to create one addressable Hermes transcript, add a verified turn, branch it, and then close or delete it deliberately. The procedure keeps transcript identity, AIP execution identity, and optional memory scope separate throug

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/hermes-agent/guides/session-lifecycle.md

Session chat can call tools and change external systems. A timeout, failed response, or missing assistant message does not prove that no effect occurred. Reconcile the transcript and downstream systems before submitting new work.

This guide uses synchronous session chat. Use the streaming guide when the application needs incremental deltas or cancellation.

Keep the identities separate

Persist these values in application state before each consequential call.

Identity	Owner	Purpose
Endpoint-qualified capability ID	AIP deployment	Selects the admitted Hermes endpoint and operation
Hermes `session_id`	Endpoint-local SessionDB	Addresses one persisted transcript row
AIP Action ID	AIP runtime	Identifies one capability invocation and its lifecycle
AIP idempotency key	AIP runtime	Fences create, patch, fork, or delete within the capability scope
Optional `session_key`	Hermes long-term memory	Selects a memory scope; it does not address the transcript

A Hermes transcript is not an AIP Session. An AIP Session carries protocol-level interaction context, while the path `session_id` selects a row and messages in the selected Hermes endpoint's `state.db`.

Do not infer tenant ownership from a `session_id`. The endpoint Bearer credential authenticates the Hermes API request, but the pinned provider does not compare the AIP principal or tenant with a row owner.

Before you start

Confirm all of these conditions:

- the intended Hermes endpoint is admitted for the current tenant;
- the requester can discover and invoke the required session capabilities;
- the endpoint and its `state.db` are isolated at the intended tenant boundary;
- the application can retain Action IDs, keys, session IDs, and redacted

results durably;

- the operator understands that model turns can create external effects;
- a requester-different, tenant-authorized reviewer is available before

`session_delete`.

The examples use `primary` as the endpoint ID. Replace it with the exact normalized ID returned by capability discovery.

1. Discover the session capabilities

Query the endpoint-qualified family before allocating a transcript:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  capability list "$AIP_URL" \
  --text 'cap:hermes_agent:primary:session' \
  --limit 20
```

Verify that the returned IDs use one endpoint and include the operations needed for the planned lifecycle. This guide uses:

```
SH
export HERMES_SESSIONS_LIST='cap:hermes_agent:primary:sessions_list'
export HERMES_SESSION_CREATE='cap:hermes_agent:primary:session_create'
export HERMES_SESSION_GET='cap:hermes_agent:primary:session_get'
export HERMES_SESSION_PATCH='cap:hermes_agent:primary:session_patch'
export HERMES_SESSION_MESSAGES='cap:hermes_agent:primary:session_messages'
export HERMES_SESSION_FORK='cap:hermes_agent:primary:session_fork'
export HERMES_SESSION_CHAT='cap:hermes_agent:primary:session_chat'
export HERMES_SESSION_DELETE='cap:hermes_agent:primary:session_delete'
```

Do not combine capabilities from endpoints that have different local session databases.

2. Allocate the transcript and Action identities

Choose an application-owned transcript ID and allocate the create Action ID and required idempotency key before submission:

```
SH
export HERMES_SESSION_ID='support-case-example-2030'
export CREATE_ACTION_ID='act_0123456789abcdef0123456789abcdef'
export CREATE_KEY='session-create-support-case-example-2030'
```

A create ID must be non-empty after trimming, no longer than 256 characters, and free of control characters, `..`, slash, backslash, or a Windows drive prefix. An explicit, stable ID is easier to reconcile than a provider-generated one.

Use a new idempotency key whenever the endpoint, session ID, title, model, system prompt, or business intent changes. Within its retention window, an exact key replay returns the original AIP result; it does not compare a changed input with the retained request.

3. Create the transcript

Create `session-create.json`:

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

```
}
```

Submit the mutation with the identities allocated above:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_CREATE" \
  --action-id "$CREATE_ACTION_ID" \
  --idempotency-key "$CREATE_KEY" \
  --mode sync \
  --input @session-create.json
```

Retain the returned `output.session.id`. A successful create reports a safe projection; it does not return the stored system prompt or model configuration. The pinned session-chat handler does not restore that stored prompt as the turn's instruction. Supply and govern any ephemeral `system_message` on the specific chat Action.

If the call ends with a transport error, timeout, remote `5xx`, or another uncertain result, do not create a replacement ID. Query the original Action, then get and list the intended transcript before deciding whether any new mutation is safe.

4. Verify the exact row and list projection

Read the exact row:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_GET" \
  --mode sync \
  --input '{"session_id":"support-case-example-2030"}'
```

Verify the returned ID, `source: "api_server"`, title, model, open state, and the expected `has_system_prompt` marker. Then locate it in the endpoint list:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSIONS_LIST" \
  --mode sync \
  --input '{"source":"api_server","limit":50,"offset":0}'
```

The list uses offset pagination over changing last-activity order. De-duplicate by ID and do not treat one page as a snapshot or ownership inventory.

5. Submit one bounded session turn

Allocate a new AIP Action ID for the turn:

```
SH
export CHAT_ACTION_ID='act_abcdef0123456789abcdef0123456789'
```

Create `session-chat.json`:

```
JSON
{
  "session_id": "support-case-example-2030",
  "message": "Summarize the verified facts already present in this transcript.",
  "system_message": "Return at most three factual bullets. Do not call tools."
}
```

Invoke synchronous session chat:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
```

```
  action call "$AIP_URL" \
  "$HERMES_SESSION_CHAT" \
  --action-id "$CHAT_ACTION_ID" \
  --mode sync \
  --input @session-chat.json
```

The contract does not require an idempotency key for chat and declares replay unsafe. Do not resend this turn automatically after an uncertain result. Read the Action status, transcript, and any systems the model could have changed.

The provider may fail to load prior history and continue with empty history. It can also return a successful model result after a transcript persistence failure. Treat the chat response as model output, not proof of history loading or durable append.

6. Verify the effective transcript and messages

First, retain `output.session_id` from the completed chat result. Context compression can move the turn to a continuation transcript and return a different effective ID.

Read messages using the last verified ID:

```
SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_MESSAGES" \
  --mode sync \
  --input '{"session_id":"support-case-example-2030"}'
```

Always inspect the top-level `output.session_id` in this result. The messages route resolves a compression continuation before it reads active messages. Update the application's current transcript ID when that value differs.

Verify, in order:

1. the user turn is present under the effective transcript;
2. the assistant row and finish metadata are present when expected;
3. the returned sequence belongs to the intended endpoint and case;
4. any claimed tool or external effect is confirmed by its authoritative

system rather than by assistant text.

The route returns the active messages of the resolved row. It does not merge every compression ancestor into one response.

7. Patch client-visible metadata

Allocate a fresh Action ID and required key, then create `session-patch.json`:

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

SH
export PATCH_ACTION_ID='act_11111111111111111111111111111111'
export PATCH_KEY='session-patch-support-case-example-2030-reviewed'

cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_PATCH" \
  --action-id "$PATCH_ACTION_ID" \
```

```
  --idempotency-key "$PATCH_KEY" \
  --mode sync \
  --input @session-patch.json
```

Patch supports only `title` and `end_reason`. Unknown fields fail instead of changing hidden provider state. Read the exact row after the mutation.

8. Fork a verified transcript

Fork only from the current, verified transcript ID. Allocate a new child ID, Action ID, and idempotency key:

```
SH
export HERMES_CHILD_ID='support-case-example-2030-alternative'
export FORK_ACTION_ID='act_22222222222222222222222222222222'
export FORK_KEY='session-fork-support-case-example-2030-alternative'
```

Create `session-fork.json`:

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

SH
cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_FORK" \
  --action-id "$FORK_ACTION_ID" \
  --idempotency-key "$FORK_KEY" \
  --mode sync \
  --input @session-fork.json
```

The provider ends an open source as `branched`, creates the child, copies the source's active messages, and only then validates the child title. Fork is not atomic. A provider `400 invalid_title` can therefore follow durable changes.

After every fork result, including an apparent rejection:

1. get the source and intended child IDs separately;
2. inspect the source `end_reason` and child `parent_session_id`;
3. read the child's active messages;
4. retain both records before deciding whether a new mutation is needed.

Do not blindly retry a failed fork. A replayed key can return the stored error without discovering or repairing a child that already exists.

9. Continue from the verified child

Use the child only after its row, lineage, and copied messages are confirmed. Submit a new chat Action whose `session_id` is the child ID, then repeat the message verification from step 6.

The parent and child now have separate transcript identities. An optional `session_key` can still point both calls at one long-term-memory scope, so do not use it when the branches must also be isolated at that layer.

10. Choose close or delete

Use metadata closure when the row and transcript must remain available:

```
JSON
{
  "session_id": "support-case-example-2030-alternative",
  "body": {
    "end_reason": "client_closed"
  }
}
```

Submit it through `session_patch` with a new Action ID and key. The first non-empty close reason wins. Closure is metadata: the pinned chat handler does not reject an ended row, so the application must enforce its own no-further-chat rule.

Use `session_delete` only when the selected row and its active messages should be removed from SessionDB. Delete is high risk and first returns an AIP `pending_approval`. Review the exact endpoint, session ID, input snapshot, policy hash, and retained lineage before an authorized reviewer decides it.

Create `session-delete.json` with the last verified target:

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

Allocate a new Action ID and required key, then submit the delete once:

```
SH
export DELETE_ACTION_ID='act_33333333333333333333333333333333'
export DELETE_KEY='session-delete-support-case-example-2030-alternative'

cargo run --locked -p aipctl -- \
  --native-bearer-token-file "$AIP_REQUESTER_TOKEN_FILE" \
  action call "$AIP_URL" \
  "$HERMES_SESSION_DELETE" \
  --action-id "$DELETE_ACTION_ID" \
  --idempotency-key "$DELETE_KEY" \
  --mode sync \
  --input @session-delete.json
```

Decide the returned approval record instead of submitting delete again. An approved decision resumes the same queued Action. Query that Action ID and then read the target and known children to verify the final state.

Deletion does not compensate model, tool, message, code, or network effects. It also does not delete long-term memory or separate transcript artifacts. The provider preserves branch and compression children by clearing their parent link, while delegate descendants are cascade-deleted.

Recover by lifecycle phase

Observation	Safe next action
Create result lost	Query the original Action, then get and list the intended ID
Chat result lost	Query the Action, read messages, and reconcile downstream effects
Returned session ID changed	Adopt the verified effective ID before the next turn or fork
`404 session_not_found`	Verify endpoint routing, then list and resolve a continuation
Fork returned `400` or delivery failed	Inspect both source and intended child; do not replay blindly
Patch result uncertain	Get the exact row and compare title or close metadata
Delete waits for approval	Review and decide the existing approval; do not resubmit delete
Delete result uncertain	Query the Action and get the row before any replacement mutation
`503 session_db_unavailable`	Restore endpoint storage, then reconcile every mutation identity

Provider SQLite is local to the configured Hermes home. A row missing from one endpoint is not proof that it never existed on another endpoint or before local storage loss.

Verify the client workflow

Before accepting the integration, use controlled data to verify that it:

1. stores endpoint ID, transcript ID, Action ID, and idempotency key separately;
2. creates a safe explicit transcript ID with a fresh required key;
3. confirms the exact row and list projection after create;
4. never treats `session_key` as transcript identity or authorization;
5. does not replay a chat turn after an uncertain outcome;
6. reads active messages after every consequential turn;
7. adopts the returned effective ID after compression rotation;
8. allows only title and close-reason metadata through patch;
9. reconciles both source and child after every fork result;
10. prevents application chat after metadata closure when policy requires it;
11. sends delete through the existing AIP approval record;
12. verifies preserved children and external effects after delete;
13. redacts transcript, reasoning, tool, and credential data from evidence;
14. records source revision and endpoint identity with the result.

This checklist is code-aligned integration evidence. It is not live endpoint, storage-durability, model, tool, or production qualification.

Related documentation

- Persistent session capabilities
- Stream chat and responses
- Authentication and endpoints
- Actions and sessions
- Approvals and policy
- Errors and recovery