Sessions and memory

Session Tools

Goal: small, hard-to-misuse tool set so agents can list sessions, fetch history, and send to another session.

Tool Names

- `sessions_list`
- `sessions_history`
- `sessions_send`
- `sessions_spawn`

Key Model

- Main direct chat bucket is always the literal key `"main"` (resolved to the current agent's main key).
- Group chats use `agent:<agentId>:<channel>:group:<id>` or `agent:<agentId>:<channel>:channel:<id>` (pass the full key).
- Cron jobs use `cron:<job.id>` .
- Hooks use `hook:<uuid>` unless explicitly set.
- Node sessions use `node-<nodeId>` unless explicitly set.

`global` and `unknown` are reserved values and are never listed. If `session.scope = "global"` , we alias it to `main` for all tools so callers never see `global` .

sessions_list

List sessions as an array of rows.

Parameters:

- `kinds?: string[]` filter: any of `"main" | "group" | "cron" | "hook" | "node" | "other"`
- `limit?: number` max rows (default: server default, clamp e.g. 200)
- `activeMinutes?: number` only sessions updated within N minutes
- `messageLimit?: number` 0 = no messages (default 0); >0 = include last N messages

Behavior:

- `messageLimit > 0` fetches `chat.history` per session and includes the last N messages.
- Tool results are filtered out in list output; use `sessions_history` for tool messages.
- When running in a **sandboxed** agent session, session tools default to **spawned-only visibility** (see below).

Row shape (JSON):

- `key` : session key (string)
- `kind` : `main | group | cron | hook | node | other`
- `channel` : `whatsapp | telegram | discord | signal | imessage | webchat | internal | unknown`
- `displayName` (group display label if available)
- `updatedAt` (ms)
- `sessionId`
- `model` , `contextTokens` , `totalTokens`
- `thinkingLevel` , `verboseLevel` , `systemSent` , `abortedLastRun`

- `sendPolicy` (session override if set)
- `lastChannel` , `lastTo`
- `deliveryContext` (normalized `{ channel, to, accountId }` when available)
- `transcriptPath` (best-effort path derived from store dir + sessionId)
- `messages?` (only when `messageLimit > 0`)

sessions_history

Fetch transcript for one session.

Parameters:

- `sessionKey` (required; accepts session key or `sessionId` from `sessions_list`)
- `limit?: number` max messages (server clamps)
- `includeTools?: boolean` (default false)

Behavior:

- `includeTools=false` filters `role: "toolResult"` messages.

  Returns messages array in the raw transcript format.

  When given a `sessionId` , OpenClaw resolves it to the corresponding session key (missing ids error).

sessions_send

Send a message into another session.

Parameters:

- `sessionKey` (required; accepts session key or `sessionId` from `sessions_list`)
- `message` (required)

`timeoutSeconds?: number` (default >0; 0 = fire-and-forget)

Behavior:

- `timeoutSeconds = 0` : enqueue and return `{ runId, status: "accepted" }` .
- `timeoutSeconds > 0` : wait up to N seconds for completion, then return `{ runId, status: "ok", reply }` .
- If wait times out: `{ runId, status: "timeout", error }` . Run continues; call `sessions_history` later.
- If the run fails: `{ runId, status: "error", error }` .
- Announce delivery runs after the primary run completes and is best-effort; `status: "ok"` does not guarantee the announce was delivered.
- Waits via gateway `agent.wait` (server-side) so reconnects don't drop the wait.
- Agent-to-agent message context is injected for the primary run.
- Inter-session messages are persisted with `message.provenance.kind = "inter_session"` so transcript readers can distinguish routed agent instructions from external user input.
- After the primary run completes, OpenClaw runs a **reply-back loop**:
  - Round 2+ alternates between requester and target agents.
  - Reply exactly `REPLY_SKIP` to stop the ping-pong.
  - Max turns is `session.agentToAgent.maxPingPongTurns` (0–5, default 5).
- Once the loop ends, OpenClaw runs the **agent-to-agent announce step** (target agent only):
  - Reply exactly `ANNOUNCE_SKIP` to stay silent.
  - Any other reply is sent to the target channel.
  - Announce step includes the original request + round-1 reply + latest ping-pong reply.

Channel Field

For groups, `channel` is the channel recorded on the session entry.

For direct chats, `channel` maps from `lastChannel`.

For cron/hook/node, `channel` is `internal`.

If missing, `channel` is `unknown`.

Security / Send Policy

Policy-based blocking by channel/chat type (not per session id).

```
{
  "session": {
    "sendPolicy": {
      "rules": [
        {
          "match": { "channel": "discord", "chatType": "group" },
          "action": "deny"
        }
      ],
      "default": "allow"
    }
  }
}
```

Runtime override (per session entry):

`sendPolicy: "allow" | "deny"` (unset = inherit config)

Settable via `sessions.patch` or owner-only `/send on|off|inherit` (standalone message).

Enforcement points:

`chat.send` / `agent` (gateway)

auto-reply delivery logic

sessions_spawn

Spawn a sub-agent run in an isolated session and announce the result back to the requester chat channel.

Parameters:

- `task` (required)
- `label?` (optional; used for logs/UI)
- `agentId?` (optional; spawn under another agent id if allowed)
- `model?` (optional; overrides the sub-agent model; invalid values error)
- `runTimeoutSeconds?` (default 0; when set, aborts the sub-agent run after N seconds)
- `cleanup?` (`delete|keep`, default `keep`)

Allowlist:

- `agents.list[].subagents.allowAgents`: list of agent ids allowed via `agentId` (`["*"]` to allow any). Default: only the requester agent.

Discovery:

Use `agents_list` to discover which agent ids are allowed for `sessions_spawn`.

Behavior:

- Starts a new `agent:<agentId>:subagent:<uuid>` session with `deliver: false`.
- Sub-agents default to the full tool set **minus session tools** (configurable via `tools.subagents.tools`).
- Sub-agents are not allowed to call `sessions_spawn` (no sub-agent → sub-agent spawning).

- Always non-blocking: returns `{ status: "accepted", runId, childSessionKey }` immediately.
- After completion, OpenClaw runs a sub-agent **announce step** and posts the result to the requester chat channel.
  - If the assistant final reply is empty, the latest `toolResult` from sub-agent history is included as `Result`.
  - Reply exactly `ANNOUNCE_SKIP` during the announce step to stay silent.
  - Announce replies are normalized to `Status` / `Result` / `Notes`; `Status` comes from runtime outcome (not model text).
  - Sub-agent sessions are auto-archived after `agents.defaults.subagents.archiveAfterMinutes` (default: 60).
  - Announce replies include a stats line (runtime, tokens, sessionKey/sessionId, transcript path, and optional cost).

Sandbox Session Visibility

Session tools can be scoped to reduce cross-session access.

Default behavior:

- `tools.sessions.visibility` defaults to `tree` (current session + spawned subagent sessions).
- For sandboxed sessions, `agents.defaults.sandbox.sessionToolsVisibility` can hard-clamp visibility.

Config:

```
{
  tools: {
    sessions: {
      // "self" | "tree" | "agent" | "all"
      // default: "tree"
      visibility: "tree",
    },
  },
  agents: {
    defaults: {
      sandbox: {
        // default: "spawned"
        sessionToolsVisibility: "spawned", // or "all"
      },
    },
  },
}
```

Notes:

- `self` : only the current session key.
- `tree` : current session + sessions spawned by the current session.
- `agent` : any session belonging to the current agent id.
- `all` : any session (cross-agent access still requires `tools.agentToAgent`).

When a session is sandboxed and `sessionToolsVisibility="spawned"` , OpenClaw clamps visibility to `tree` even if you set `tools.sessions.visibility="all"` .

‹ Session Pruning Memory ›