Fundamentals

Agent Loop

An agentic loop is the full “real” run of an agent: intake → context assembly → model inference → tool execution → streaming replies → persistence. It’s the authoritative path that turns a message into actions and a final reply, while keeping session state consistent.

In OpenClaw, a loop is a single, serialized run per session that emits lifecycle and stream events as the model thinks, calls tools, and streams output. This doc explains how that authentic loop is wired end-to-end.

Entry points

- Gateway RPC: `agent` and `agent.wait`.
- CLI: `agent` command.

How it works (high-level)

1. `agent` RPC validates params, resolves session (sessionKey/sessionId), persists session metadata, returns `{ runId, acceptedAt }` immediately.
2. `agentCommand` runs the agent:
   - resolves model + thinking/verbose defaults
   - loads skills snapshot

calls `runEmbeddedPiAgent` (pi-agent-core runtime)

emits **lifecycle end/error** if the embedded loop does not emit one

3. `runEmbeddedPiAgent` :

   serializes runs via per-session + global queues

   resolves model + auth profile and builds the pi session

   subscribes to pi events and streams assistant/tool deltas

   enforces timeout → aborts run if exceeded

   returns payloads + usage metadata

4. `subscribeEmbeddedPiSession` bridges pi-agent-core events to OpenClaw `agent` stream:

   tool events ⇒ `stream: "tool"`

   assistant deltas ⇒ `stream: "assistant"`

   lifecycle events ⇒ `stream: "lifecycle"` (`phase: "start" | "end" | "error"`)

5. `agent.wait` uses `waitForAgentJob` :

   waits for **lifecycle end/error** for `runId`

   returns `{ status: ok|error|timeout, startedAt, endedAt, error? }`

Queueing + concurrency

Runs are serialized per session key (session lane) and optionally through a global lane.

This prevents tool/session races and keeps session history consistent.

Messaging channels can choose queue modes (collect/steer/followup) that feed this lane system. See **Command Queue**.

Session + workspace preparation

- Workspace is resolved and created; sandboxed runs may redirect to a sandbox workspace root.
- Skills are loaded (or reused from a snapshot) and injected into env and prompt.
- Bootstrap/context files are resolved and injected into the system prompt report.
- A session write lock is acquired; `SessionManager` is opened and prepared before streaming.

Prompt assembly + system prompt

- System prompt is built from OpenClaw's base prompt, skills prompt, bootstrap context, and per-run overrides.
- Model-specific limits and compaction reserve tokens are enforced.
- See System prompt for what the model sees.

Hook points (where you can intercept)

OpenClaw has two hook systems:

- **Internal hooks** (Gateway hooks): event-driven scripts for commands and lifecycle events.
- **Plugin hooks**: extension points inside the agent/tool lifecycle and gateway pipeline.

Internal hooks (Gateway hooks)

- `agent:bootstrap` : runs while building bootstrap files before the system prompt is finalized. Use this to add/remove bootstrap context files.

Command hooks: `/new`, `/reset`, `/stop`, and other command events (see Hooks doc).

See **Hooks** for setup and examples.

Plugin hooks (agent + gateway lifecycle)

These run inside the agent loop or gateway pipeline:

- `before_model_resolve`: runs pre-session (no `messages`) to deterministically override provider/model before model resolution.
- `before_prompt_build`: runs after session load (with `messages`) to inject `prependContext` / `systemPrompt` before prompt submission.
- `before_agent_start`: legacy compatibility hook that may run in either phase; prefer the explicit hooks above.
- `agent_end`: inspect the final message list and run metadata after completion.
- `before_compaction` / `after_compaction`: observe or annotate compaction cycles.
- `before_tool_call` / `after_tool_call`: intercept tool params/results.
- `tool_result_persist`: synchronously transform tool results before they are written to the session transcript.
- `message_received` / `message_sending` / `message_sent`: inbound + outbound message hooks.
- `session_start` / `session_end`: session lifecycle boundaries.
- `gateway_start` / `gateway_stop`: gateway lifecycle events.

See **Plugins** for the hook API and registration details.

Streaming + partial replies

Assistant deltas are streamed from pi-agent-core and emitted as `assistant` events.

Block streaming can emit partial replies either on `text_end` or `message_end`.

Reasoning streaming can be emitted as a separate stream or as block replies.

See **Streaming** for chunking and block reply behavior.

Tool execution + messaging tools

Tool start/update/end events are emitted on the `tool` stream.

Tool results are sanitized for size and image payloads before logging/emitting.

Messaging tool sends are tracked to suppress duplicate assistant confirmations.

Reply shaping + suppression

Final payloads are assembled from:

- assistant text (and optional reasoning)
- inline tool summaries (when verbose + allowed)
- assistant error text when the model errors

`NO_REPLY` is treated as a silent token and filtered from outgoing payloads.

Messaging tool duplicates are removed from the final payload list.

If no renderable payloads remain and a tool errored, a fallback tool error reply is emitted (unless a messaging tool already sent a user-visible reply).

Compaction + retries

Auto-compaction emits `compaction` stream events and can trigger a retry. ›

On retry, in-memory buffers and tool summaries are reset to avoid duplicate output.

See **Compaction** for the compaction pipeline.

Event streams (today)

`lifecycle` : emitted by `subscribeEmbeddedPiSession` (and as a fallback by `agentCommand`)

`assistant` : streamed deltas from pi-agent-core

`tool` : streamed tool events from pi-agent-core

Chat channel handling

Assistant deltas are buffered into chat `delta` messages.

A chat `final` is emitted on **lifecycle end/error**.

Timeouts

`agent.wait` default: 30s (just the wait). `timeoutMs` param overrides.

Agent runtime: `agents.defaults.timeoutSeconds` default 600s; enforced in `runEmbeddedPiAgent` abort timer.

Where things can end early

Agent timeout (abort)

AbortSignal (cancel)

Gateway disconnect or RPC timeout

`agent.wait` timeout (wait-only, does not stop agent)

‹ Agent Runtime › System Prompt ›