ChatCompress Action Reference

Reference for the `ChatCompress` Display action — summarises the current conversation transcript via one LLM call and atomically replaces it with a single synthetic message, preserving semantic continuity while reducing token cost.

Purpose

`ChatCompress` is a Display Action that condenses a long conversation history into a single compact summary. It makes exactly one LLM call — passing the full transcript plus a system-level summarisation prompt — then atomically replaces the in-memory transcript with a single synthetic assistant-role message whose body is the LLM-produced summary. Subsequent `ChatRequest` turns see only that summary as their prior context.

Use it when a conversation has grown long enough to push against the model's context window, or when you want to reduce token consumption on follow-up turns without discarding conversational context entirely.

When to use

- **Long conversations** — after many turns, transcript length grows. Compress before the model's context window fills up and earlier turns start getting dropped silently.
- **Shift continuity** — compress at shift change to produce a concise handover summary the next operator can read, while giving the model a compact context baseline.
- **Token cost control** — on hosted or metered models, compressing a 50-turn transcript to a single summary message substantially reduces per-query token spend.
- **After a topic switch** — compress to carry a "what we discussed" baseline without dragging the full prior turn-by-turn exchange into the new topic.

> If you want to discard context entirely rather than summarise it, use ChatClear Action Reference instead. `ChatClear` makes no LLM call and costs nothing beyond the cache-drop operation.

How it works

When fired, `ChatCompress` executes the following sequence for the resolved `(clientGuid, chatName)` pair:

1. The target `TChatSession` control is resolved via the **Object** field (explicit name) or a visual-tree walk (auto, if Object is empty).
2. **Short-circuit check:** if the transcript contains fewer than 2 messages, the action returns `status="ok"` immediately without calling the LLM. Compressing a one-message or empty transcript is a no-op.
3. The full transcript is serialised and sent to the LLM with a system-level summarisation prompt asking for a concise synthesis of the conversation.
4. On a successful LLM response, the transcript is **atomically replaced** with a single synthetic message (role: assistant) whose body is the summary text.
5. On LLM failure, timeout, or empty summary, the original transcript is **preserved unchanged** and the action returns `status="error"`. No partial replacement occurs.

The next `ChatRequest` turn sees only the single summary message as prior context. The full original turn history is not recoverable after a successful compress — use `ChatClear` if you need a reversible reset.

Configuration

On a Button (or any clickable control), add an **Action** dynamic with:

Field	Setting
Action type	`ChatCompress`
Object	Name of the target `TChatSession` control on the same Display. Leave empty to auto-resolve to the first ChatSession control found in the visual tree.
Return	Optional tag that receives the reply envelope JSON. When `status="ok"`, the `text` field carries the LLM-produced summary. Useful for displaying a "Summary" badge or logging the compress result.
Result 1, Result 2, … (optional)	Tags computed from the reply via Expressions — for example, `@Tag.CompressSummary.JsonString("text")` to surface the summary text in a label.
Query	Not used by `ChatCompress`. The action takes no user input; the transcript itself is the input.

The Action editor hides fields that do not apply to `ChatCompress`. The Query field is hidden; Object, Return, and Result/Expression rows surface.

Reply envelope

The reply envelope follows the same JSON schema as `ChatRequest` and `AI.Execute` — see Local AI Reply Envelope Schema for the full field reference. Key fields for `ChatCompress`:

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "text":      "<LLM-produced summary on success, or '' on error>",
  "status":    "ok | error | disabled",
  "toolTrace": [],
  "latencyMs": 1840,
  "warnings":  []
}
```

On success (`status="ok"`), `text` carries the summary that was written into the transcript as the replacement message. On failure (`status="error"`), `text` is empty and the original transcript is intact. `toolTrace` is always empty — compress does not dispatch platform tools. `latencyMs` reflects the LLM round-trip for the summarisation call.

Short-circuit cases

- **Empty transcript** (0 messages) — returns `status="ok"`, no LLM call, transcript unchanged.
- **Single message** (1 message) — returns `status="ok"`, no LLM call, transcript unchanged. A single message cannot be meaningfully summarised further.

Both short-circuit cases return immediately without calling the LLM, so they are indistinguishable from a successful compress in terms of the return envelope. The `text` field is empty for short-circuit returns.

Gates

`ChatCompress` checks a single gate before executing:

- **`SolutionCapabilities[LocalAI].Enabled` must be `true`** — the master Local AI kill-switch. When disabled, `ChatCompress` returns `status="disabled"` without calling the LLM or touching the transcript.

`ChatCompress` does *not* inspect `ModelOptions` tool-surface bits (`EnableChatHistory`, `EnableRuntimeMCP`, per-category sub-bits). Transcript management is independent of the tool-surface configuration; the compress call uses the LLM solely for summarisation, not for tool dispatch.

Wall-clock budget

`ChatCompress` is subject to the same 60-second wall-clock timeout as `ChatRequest`. For very long transcripts, the summarisation POST may itself take several seconds. If the LLM does not respond within the budget, the action returns `status="error"` and the original transcript is preserved.

> ⚠ The LLM is expected to run on a **separate GPU-equipped host**, not on the TServer machine itself. On a GPU host, compressing a 20–30 turn transcript typically takes 1–3 seconds. On a CPU-only host (the degraded fallback — ~2–4 tokens/sec), the same compress call typically takes 3–8 seconds or more. Run a quick test at your hardware tier before exposing a “Compress” button to operators who may expect an immediate response.

Target resolution

`ChatCompress` resolves the target chat session using the same two-path logic as `ChatClear`:

Path	When used	Behavior
Path A — explicit name	`Object` field is set to a non-empty value	The platform resolves the named element on the active Display panel. If no control with that name exists, the action returns an error envelope.
Path B — visual-tree walk	`Object` field is empty	The platform walks the visual tree of the active Display and targets the first `TChatSession` control found. Sufficient for Displays with a single chat control. On Displays with multiple ChatSession controls, always use Path A to avoid ambiguity.

See also

- ChatClear Action Reference — discards the transcript entirely without an LLM call; use when continuity is not needed.
- ChatRequest Action Reference — the primary chat action that sends operator queries to the LLM and manages the transcript.
- ChatSession Control Reference — the Display control that renders the conversation thread.
- Local AI — the parent section, includes a step-by-step quick-start.
- Local AI Reply Envelope Schema — full reply envelope schema.

In this section...