Claude Code Message Threads: What Happens After 5 Minutes?

TechRounder PDF Edition

Live article: https://www.techrounder.com/ai/claude-code-message-threads-five-minute-expiry/

By Vipin PG | Published November 4, 2026 | Updated November 4, 2026 | Format: Deep Dive | 11 min read

In brief

Anthropic introduced message threads to reduce the amount of conversation history that Claude Code and other agents need to upload with each API request. After the first request, the client can send new messages while Anthropic temporarily retains the earlier context.

Anthropic introduced message threads to reduce the amount of conversation history that Claude Code and other agents need to upload with each API request. After the first request, the client can send new messages while Anthropic temporarily retains the earlier context.

The server-side thread expires after five minutes of inactivity. That can happen while you review code, run tests, or wait to approve an action.

When a thread expires, the client can send its saved conversation again and start a new thread. The next response may take longer because the full history needs uploading. Expiry alone does not delete the client's conversation or add a special token charge.

The details depend on what the client saves, how it handles failed continuations, and whether prompt caching is also enabled.

What are Claude Code message threads?

Message threads are a new feature of Anthropic's Messages API, released in public beta on November 2, 2026. The date is confirmed in the Claude Platform release notes and Anthropic's launch announcement.

A coding assistant normally sends the conversation history with each Messages API request. In a long session, that can include previous prompts, code, file contents and tool results, even if the next user message is short.

The first threaded request includes the necessary history and configuration. Later requests carry the new messages and a reference to the previous response. Anthropic's API reconstructs the conversation before passing it to Claude.

The smaller upload leaves the model's context unchanged. Claude still processes the reconstructed history, and the normal context-window and token-accounting rules continue to apply.

Image: Figure 1: What changes when an application uses message threads?

Why did Anthropic introduce message threads?

A coding agent may make many API requests while working through a task. Re-uploading an expanding transcript adds network and request-processing overhead before Claude can begin responding.

Anthropic reports about 35% lower median time to first token (TTFT) for Claude Code requests with 500,000 to 1 million tokens of context. TTFT measures how soon the response starts. It does not include all the time spent generating code, running tests or waiting for tools.

An Anthropic developer-account follow-up also reported a reduction in median request payload from roughly 800 KB to 1.8 KB in long conversations. These are Anthropic's own measurements. The reviewed material does not provide an independent reproducible benchmark.

The potential gain is greatest when an agent sends large amounts of repeated context, such as code, screenshots or document content. A short conversation produces much less upload overhead to eliminate.

What happens when a Claude message thread expires after five minutes?

According to Anthropic's message threads documentation, a thread expires five minutes after its last request. Each response that the API successfully stores restarts the timer.

For example, Claude replies at 10:00 AM. You inspect a proposed change, run a build and return at 10:07 AM. The stored thread may have expired during that pause.

A continuation using the previous response ID can then return HTTP '404' with 'thread_not_found'. The client can recover by creating a new thread from the conversation history it kept locally.

Image: Figure 2: A seven-minute pause and the recovery process

Does Claude lose the conversation?

Claude Code and other clients can maintain their own transcripts independently of Anthropic's temporary server-side thread. If that transcript is intact, the client has the material it needs to resume the conversation.

A custom client that saves only the response ID cannot recover the full history through a thread retrieval endpoint. Anthropic provides no such endpoint, so that client could lose the context needed for replay.

Will the developer need to restart Claude Code?

Claude Code users should not normally need to restart the program or paste old messages when a thread expires. Anthropic says threads are enabled for most Claude Code users. Its SDK helper also provides automatic replay, although exact behavior depends on the client and its configuration.

What if a build, test, or approval takes longer than five minutes?

A six-minute build or an extended human approval can leave the thread idle. When the result arrives, a client with a stored transcript can create a new thread that includes the completed tool result or approval. Replaying the record of an earlier command does not itself execute the command again.

Applications that perform external actions should separately preserve tool results and approval state. Their retry logic must prevent accidental duplicate actions after interruptions.

Does Anthropic keep the thread for a full hour?

Anthropic gives two deadlines. Continuation expires after five minutes, while stored thread data is deleted no more than one hour after its last use. The one-hour retention limit does not extend the period in which a thread can be continued.

How does recovery work behind the scenes?

The protocol uses three operations:

1. Create: The client sends the conversation, system instructions and tools, plus ‘"thread": {"type": "create"}‘ . The response includes an ID.

2. Continue: The client supplies only the new messages and ‘"thread": {"type": "continue", "previous_message_id": "..."}‘ . The next response provides another ID.

3. Replay: If continuation is unavailable, the client sends the complete saved conversation, including the latest message, using a new create request.

Image: Figure 3: Recovery when a thread cannot be continued

Storage is best effort. A successful response does not confirm that Anthropic saved the turn for later continuation. The client learns whether that response can be continued only when it sends the next request.

The SDK thread helper holds conversation history in memory and attempts one replay for certain recoverable errors. Applications that need to survive a restart must persist the transcript elsewhere.

Message threads vs. prompt caching: what is the difference?

Both features commonly use five-minute lifetimes, but they manage different kinds of state.

Message threads let the client upload fewer bytes on later requests. Prompt caching allows eligible prompt prefixes to be reused and affects how cache reads and writes are billed. Anthropic supports using the two features together.

Question: What is saved? | Message threads: Temporary conversation state for later API continuation | Prompt caching: Reusable processed prompt prefixes

Question: Main benefit | Message threads: Smaller requests and less preparation overhead | Prompt caching: Faster reuse of cached context and cache-specific billing

Question: Default lifetime | Message threads: Five-minute continuation expiry | Prompt caching: Five-minute cache option; one-hour option also documented

Question: Does it reduce model context size? | Message threads: No | Prompt caching: No

Question: Does it automatically reduce token charges? | Message threads: No | Prompt caching: Eligible cache reads can cost less

Question: Can it work alongside the other feature? | Message threads: Yes | Prompt caching: Yes

Suppose a thread expires while a separately configured one-hour prompt cache remains valid. Recreating the thread still requires the client to send the full transcript, but the new model request may reuse the cached prefix.

A cache hit depends on matching content, eligible prefixes, breakpoints and model behavior. Anthropic explains those requirements and the rates in its prompt caching guide.

Do message threads reduce Claude Code costs or token usage?

Anthropic says message threads leave standard token pricing and rate limits unchanged. A continuation carries fewer bytes over the network, while the model still receives the full reconstructed conversation.

A documented thread error is returned before the model runs, so that failed request consumes no model tokens. A successful replay is billed normally. If a separate prompt cache also expired during the pause, the next request may cost more than an equivalent cache hit.

The same distinction applies to Claude Code usage limits: reducing uploaded data does not shrink the context being processed or increase plan allowances.

Common message thread errors and what they mean

Claude Code users will generally encounter recovery through the client. Developers building integrations must inspect the API's error code before deciding whether to replay the history.

Error code: 'thread_not_found' | Likely meaning: The referenced thread is unavailable, expired, or unusable | What to do: Replay the saved full conversation in a new thread

Error code: 'thread_fingerprint_mismatch' | Likely meaning: A request field that must stay fixed has changed | What to do: Restore compatible settings or create a new thread

Error code: 'thread_already_continued' | Likely meaning: The response ID has already been used | What to do: Use the newest valid response ID or replay

Error code: 'thread_continue_in_flight' | Likely meaning: Another continuation from that ID is underway | What to do: Allow it to complete or recover appropriately

Error code: 'thread_unsupported_request' | Likely meaning: The feature or organization setting is incompatible | What to do: Remove the unsupported combination or use unthreaded requests

For a 'thread_not_found' response, inspect 'error.details.error_code'. An HTTP 404 can also refer to a nonexistent model. Thread continuation may fail because the earlier response was not stored, is still generating, was produced by a fallback model, or belongs to a different caller identity. Anthropic lists these cases in its recovery documentation.

Using message threads in your own Claude API application

Claude Code handles supported thread requests within the client. Direct Messages API integrations need to supply the beta header 'message-threads-2026-08-12' and the appropriate 'thread' field.

These JSON snippets show only the fields relevant to threading. They are not complete API requests.

1. Create a thread

```
{
"thread": {"type": "create"},
"messages": [
{"role": "user", "content": "Review this function"}
]
}
```

The create request also needs the required model settings and any system instructions or tools. Save the returned response ID.

2. Continue using the previous response ID

```
{
"thread": {
"type": "continue",
"previous_message_id": "<previous response id>"
},
"messages": [
{"role": "user", "content": "Suggest a safer version"}
```

```
]
}
```

Send only messages added since the previous response. Do not resend the stored system or tool definitions on continuation, even as ‘null‘. Other fixed settings, including the model and certain thinking, cache and context options, must remain consistent.

3. Recover when continuation fails

When the API returns a recoverable thread error, send the complete saved transcript and the newest message or tool result in a fresh create request. Save its response ID for subsequent turns.

Anthropic provides SDK thread helpers for Python, TypeScript, C#, Go, Java, PHP and Ruby. They handle response IDs and supported recovery cases. Check the official SDK examples against your installed SDK version.

Important limitations for coding agents and API gateways

Developers need to account for the following restrictions when adding message threads to coding agents or gateways.

- Keep the complete transcript. The server does not offer an API to list, retrieve or delete stored threads. An SDK helper’s in-memory history will not survive every restart.
- A thread is append-only. A previous response ID can be continued only once. To create independent branches, start a new thread from a locally copied history. The SDK also offers a fork helper.
- Streaming requires care. Do not continue from an ID received at ‘message_start‘ until the response reaches ‘message_stop‘ . Interrupted streams need suitable retry and tool-state handling.
- Tool and MCP configuration needs attention. Stored system and tool definitions remain fixed, while certain MCP server URLs, credentials and container details must still be supplied as required. Not every gateway or proxy necessarily supports this beta protocol.
- Some API features are incompatible. The documentation excludes Message Batches, threaded token-counting calls, certain server-side compaction and context-editing operations, and configurations involving inference hooks or HIPAA readiness.
- Context management still matters. Threads do not increase the model’s context window. If the conversation needs summarization, the application must create a new thread from the revised history where supported.
- Parallel agents are separate. Message threads do not create subagents or coordinate workers. Claude Code subagents have their own contexts and requests.

See Anthropic’s unsupported features documentation, subagents guide and memory documentation for the separate behaviors of those systems.

Privacy and data retention: what gets stored?

Anthropic says it stores thread contents in encrypted server memory without writing them to durable storage, and deletes them no more than an hour after last use. Stored contents include the conversation, create-time instructions and tools, and a fingerprint of fixed request settings. These statements describe Anthropic’s published retention policy; they are not independent audit findings.

Message threads are not eligible for zero data retention (ZDR) or HIPAA-enabled use. Teams with those requirements should check eligibility before enabling them. Thread deletion also does not delete client transcripts, logs or files kept elsewhere. Anthropic describes those boundaries in its API data retention guide.

What are developers reporting so far?

In a November 3 r/ClaudeAI discussion, users asked what happens when an approval or tool call takes longer than five minutes. A November 4 thread raised similar questions about waiting for subagents. Those comments describe concerns and individual experiences, not a measured failure rate.

An October 16 Claude Code GitHub issue documents successful continuation after 200- and 270-second pauses, then ‘thread_not_found‘ after 360 seconds, followed by replay. It describes one setup and predates the November 2 public announcement.

Developers have also requested support in Hermes Agent and OpenClaw. Those requests show interest in adoption; they do not confirm that either project has released a working integration.

When are message threads actually useful?

Workflow: Long, active Claude Code sessions | Likely value: High: large histories and frequent turns | What to watch: Replay after idle periods

Workflow: Automated coding and tool loops | Likely value: High when the loop stays active | What to watch: Slow tests, tool state and retries

Workflow: Computer-use and document agents | Likely value: Potentially high for large repeated inputs | What to watch: Multimodal compatibility and storage rules

Workflow: Human-approved agent workflows | Likely value: Variable | What to watch: Approval gaps longer than five minutes

Workflow: Short, occasional API chats | Likely value: Usually limited | What to watch: Small baseline request overhead

The likely benefits in this table follow from request size and workflow timing, not independent tests. For an actual comparison, measure uploaded bytes, time to first token, total turn duration, cache hits, replay frequency and idle gaps with the same model and settings. Include both rapid agent loops and longer pauses.

Frequently asked questions

Will Claude forget my project after five minutes?

The API continuation state can expire after five minutes. Project files, the client’s saved transcript and Claude Code memory have separate lifetimes. A custom client must keep its transcript to recover a thread.

Can I extend the five-minute thread expiry?

The reviewed documentation provides no configurable thread lifetime or dedicated keepalive operation. Sending artificial model requests to keep a thread active can consume resources and is not documented as a free workaround.

Does an expired thread consume tokens when it returns an error?

Documented thread errors run no model inference and consume no model tokens. The next successful replay follows standard token and cache billing.

Does a one-hour prompt cache keep the message thread alive?

A one-hour prompt cache does not extend the thread’s five-minute continuation lifetime. It may still provide a cache hit during the replay.

Can two agents continue from the same response ID?

Each response ID can be continued once. Concurrent or repeated attempts can return ‘thread_continue_in_flight‘ or ‘thread_already_continued‘. Independent branches require separate threads.

Does the feature work with every Claude-compatible API provider?

Anthropic lists the Claude API, Claude Platform on AWS, Claude Models on Google Cloud (Anthropic managed), and Microsoft Foundry. A third-party proxy or compatible endpoint needs separate confirmation of support.

What this means for Claude Code users

Message threads reduce repeated uploads during long Claude Code sessions. Anthropic reports the biggest measured benefit for requests with 500,000 to 1 million tokens of context, where the delay before the first token can be substantial.

A pause beyond five minutes can require the client to recreate the thread from its saved transcript. That replay may delay the next response, but it does not automatically delete context, rerun completed commands or add an expiry fee.

Claude Code handles much of this through its client. Custom agent developers should keep a complete, durable transcript and preserve tool and approval state so they can recover safely when a thread cannot be continued.

References

1. platform.claude.com - docs / en - https://platform.claude.com/docs/en/release-notes/overview
2. claude.com - resources / articles - https://claude.com/resources/articles/message-threads
3. x.com - ClaudeDevs / status - https://x.com/ClaudeDevs/status/2117332451231956997
4. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/message-threads
5. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/message-threads#sdk-helper-handles
6. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/prompt-caching
7. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/message-threads#recover
8. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/message-threads#use-a-thread-with-an-sdk
9. platform.claude.com - docs / en - https://platform.claude.com/docs/en/build-with-claude/message-threads#unsupported
10. code.claude.com - docs / en - https://code.claude.com/docs/en/sub-agents
11. code.claude.com - docs / en - https://code.claude.com/docs/en/memory
12. platform.claude.com - docs / en - https://platform.claude.com/docs/en/manage-claude/api-and-data-retention
13. reddit.com - r / ClaudeAI - https://www.reddit.com/r/ClaudeAI/comments/1xmh012/anthropic_added_message_threads_to_the_messages/
14. reddit.com - r / ClaudeAI - https://www.reddit.com/r/ClaudeAI/comments/1xnc4ew/anthropic_shipped_message_threads_beta_claude/
15. github.com - anthropics / claude-code - https://github.com/anthropics/claude-code/issues/102850

16. github.com - NousResearch / hermes-agent - https://github.com/NousResearch/hermes-agent/issues/148119

17. github.com - openclaw / openclaw - https://github.com/openclaw/openclaw/issues/191987