The Transcript Abstraction: Deep Dive

> *Last updated: 2026-02-27. Based on full codebase audit + analysis of real Claude Code transcripts on this machine.*

Table of Contents

---

1. What the Abstraction Provides

The Transcript abstraction is the **richest data structure in fuel-code**. It takes the raw JSONL conversation log that Claude Code writes to disk during every session and transforms it into structured, queryable, searchable Postgres rows — then generates an LLM summary on top.

It spans two database tables, an async pipeline, crash recovery, S3 archival, a CLI renderer, and API endpoints. It is NOT one of the original five abstractions from CORE.md but emerged as first-class during Phase 2.

Core value proposition: Turn an opaque, append-only JSONL file into structured data you can query ("show me all tool uses in this session", "what was the initial prompt", "how much did this session cost").

---

2. The Three Layers

Defined in `packages/shared/src/types/transcript.ts`:

Layer 1: Raw (What the Parser Reads)

```
// The 7 line types observed in Claude Code transcripts
type TranscriptLineType =
  | "user" | "assistant" | "system" | "summary"
  | "progress" | "file-history-snapshot" | "queue-operation";

// The 4 content block types within messages
type ContentBlockType = "text" | "thinking" | "tool_use" | "tool_result";

interface RawTranscriptLine {
  type: TranscriptLineType;
  parentUuid?: string;
  isSidechain?: boolean;
  sessionId?: string;
  cwd?: string;
  version?: string;
```

```
  gitBranch?: string;
  timestamp?: string;
  uuid?: string;
  message?: {
    role?: string;
    model?: string;
    id?: string;
    content?: string | RawContentBlock[];
    usage?: TokenUsage;
    stop_reason?: string | null;
  };
  snapshot?: { timestamp?: string };
  data?: unknown;
}
```

This is a **minimal subset** of what CC actually writes to JSONL (see Section 5 for the full picture).

Layer 2: Parsed (Postgres Rows)

Two types mapping 1:1 to database tables:

`TranscriptMessage` → `transcript_messages` (21 columns):

Field	Type	Purpose
`id`	TEXT PK	ULID (generated by us, NOT CC's uuid)
`session_id`	TEXT FK	Links to sessions table
`line_number`	INTEGER	Position in original JSONL file
`ordinal`	INTEGER	Conversation-order position
`message_type`	TEXT	"user" / "assistant" / "system" / "summary"
`role`	TEXT	"human" / "assistant" / "system"
`model`	TEXT	Model that generated the response
`tokens_in` / `tokens_out`	INTEGER	Token counts
`cache_read` / `cache_write`	INTEGER	Cache token counts
`cost_usd`	NUMERIC(10,6)	Computed cost for this message
`compact_sequence`	INTEGER	Compaction epoch (always 0 — never populated)
`is_compacted`	BOOLEAN	Pre-compaction flag (always false — never populated)
`timestamp`	TIMESTAMPTZ	When the message was produced
`raw_message`	JSONB	Full raw line for lossless reconstruction

`metadata`	JSONB	Extensible metadata (always `{}`)
`has_text` / `has_thinking` / `has_tool_use` / `has_tool_result`	BOOLEAN	Denormalized flags for fast filtering

`ParsedContentBlock` → `content_blocks` (14 columns):

Field	Type	Purpose
`id`	TEXT PK	ULID
`message_id`	TEXT FK	Links to transcript_messages
`session_id`	TEXT FK	Denormalized for session-level queries
`block_order`	INTEGER	Position within the parent message
`block_type`	TEXT	"text" / "thinking" / "tool_use" / "tool_result"
`content_text`	TEXT	Text content (has GIN full-text search index)
`thinking_text`	TEXT	Thinking block content
`tool_name`	TEXT	e.g., "Read", "Edit", "Bash" (has partial index)
`tool_use_id`	TEXT	CC's tool_use_id for linking tool_use → tool_result
`tool_input`	JSONB	Tool parameters
`tool_result_id`	TEXT	Links tool_result back to its tool_use
`is_error`	BOOLEAN	Whether tool result was an error
`result_text`	TEXT	Truncated tool result (max 256KB inline)
`result_s3_key`	TEXT	S3 key for large results (NEVER USED)
`metadata`	JSONB	Extensible (used for truncation info)

Database schema: `packages/server/src/db/migrations/002_transcript_tables.sql`

Layer 3: Stats (Aggregates)

```
interface TranscriptStats {
  total_messages: number;
  user_messages: number;
  assistant_messages: number;
  tool_use_count: number;
  thinking_blocks: number;
  subagent_count: number;      // Counted by tool_name === "Task"
  tokens_in: number;
  tokens_out: number;
  cache_read_tokens: number;
  cache_write_tokens: number;
  cost_estimate_usd: number;   // Hardcoded Sonnet pricing
```

```
  duration_ms: number;         // first_timestamp to last_timestamp
  initial_prompt: string | null; // First 1000 chars of first user message
}
```

These stats are written as denormalized columns on the `sessions` table when the pipeline advances lifecycle to `parsed`.

---

3. The Four Flavors

The transcript exists in **4 distinct forms** at different points in its lifecycle:

#	Form	Location	Format	When
1	**Raw JSONL on disk**	`~/.claude/projects/{encoded-path}/{sessionId}.jsonl`	Append-only JSONL	Written by CC during session
2	**Raw JSONL in S3**	`transcripts/{canonicalId}/{sessionId}/raw.jsonl`	Same JSONL	Uploaded by CLI hook after session ends
3	**Structured in Postgres**	`transcript_messages` + `content_blocks` tables	Normalized rows	Created by session pipeline after parsing
4	**Parsed backup in S3**	`transcripts/{canonicalId}/{sessionId}/parsed.json`	Full `ParseResult` as JSON	Best-effort backup after pipeline

Additionally, subagent transcripts exist as separate files at `{sessionId}/subagents/agent-{agentId}.jsonl` — these are **never uploaded or parsed** by fuel-code.

---

4. The Creation/Modification Pipeline

4.1 Ingest Path

```
CC session ends
  → SessionEnd hook fires
    → CLI emits session.end event
    → CLI runs `fuel-code transcript upload --session-id X --file Y`
      → POST /api/sessions/:id/transcript/upload
        → Buffer body → Upload to S3 → Store s3_key on session row
        → If lifecycle === 'ended': trigger pipeline
```

Key files:

- `packages/cli/src/commands/transcript.ts` — CLI upload command (always exit 0, never throws)
- `packages/server/src/routes/transcript-upload.ts` — HTTP endpoint, idempotent, 200MB limit

4.2 Processing Pipeline

```
runSessionPipeline(deps, sessionId)
  1. Validate: session lifecycle === 'ended', has transcript_s3_key
  2. Set parse_status = 'parsing' (claim lock)
  3. Download raw JSONL from S3
  4. parseTranscript() → ParseResult { messages, contentBlocks, stats, errors,
metadata }
  5. Transaction: DELETE old rows → batch INSERT messages (500/batch) → batch INSERT
blocks
  6. transitionSession('ended' → 'parsed') + write stats to session columns
  7. generateSummary() → Claude Sonnet API call → transitionSession('parsed' →
'summarized')
  8. Upload parsed.json backup to S3 (best-effort)
```

Key files:

- `packages/core/src/transcript-parser.ts` — Pure function, two-pass parser
- `packages/core/src/session-pipeline.ts` — Orchestrator + bounded async queue
- `packages/core/src/summary-generator.ts` — Renders condensed markdown, calls Anthropic API
- `packages/core/src/summary-config.ts` — Config loader (model, temperature, max tokens)

4.3 Parser Internals

The parser (`parseTranscript()`) works in 3 passes:

Pass 1 — Classify: JSON-parse each line, skip `progress` / `file-history-snapshot` / `queue-operation` , group assistant lines by `message.id` (CC streams multi-line responses sharing the same message ID).

Pass 2 — Build: Walk classified lines in order. For each:

- Assistant: merge all lines in the `message.id` group into one `TranscriptMessage` + N `ParsedContentBlock` rows. Token usage from the LAST line (most complete).
- User/system/summary: one `TranscriptMessage` per line, extract content blocks from `message.content` .

Pass 3 — Stats: Count messages, blocks, tokens. Compute cost (hardcoded pricing). Extract initial prompt.

4.4 Summary Generation

`generateSummary()` renders the transcript into a condensed markdown prompt:

- User messages: `[User]: {text}` (first 500 chars)
- Assistant text: `[Assistant]: {text}` (first 300 chars)
- Tool uses: `- Used {tool_name}` (name only)
- Thinking blocks: **skipped entirely**
- Tool results: **skipped entirely**

- If >8000 chars: keep first 3000 + last 3000 with truncation marker

Calls Claude Sonnet 4.5 (`claude-sonnet-4-5-20250929`) with a 150-token max, temperature 0.3, 30s timeout.

4.5 Recovery

Two recovery mechanisms in `packages/core/src/session-recovery.ts` :

1. **`recoverStuckSessions()`** : Finds sessions with lifecycle=ended but parse_status=pending/parsing (stuck >10min). Re-triggers pipeline if transcript exists in S3, marks failed otherwise.
2. **`recoverUnsummarizedSessions()`** : Finds sessions at lifecycle=parsed with no summary (stuck >10min). Resets to ended and re-runs the full pipeline.

4.6 Serving

- `GET /api/sessions/:id/transcript` — Returns parsed messages with content blocks joined via `json_agg` (requires parse_status=completed)
- `GET /api/sessions/:id/transcript/raw` — Returns presigned S3 URL for raw JSONL download
- `packages/cli/src/lib/transcript-renderer.ts` — Terminal renderer with word-wrap, collapsed thinking, box-drawing tool trees

---

5. What Claude Code Transcripts Actually Contain

This section documents the **actual JSONL format** that Claude Code writes — based on Anthropic docs, community research, and inspection of real transcripts on this machine. This is what we're working with as source material.

5.1 File Locations

```
~/.claude/projects/{url-encoded-path}/
  {sessionId}.jsonl                          # Main session transcript
  {sessionId}/subagents/
    agent-{shortId}.jsonl                    # Subagent transcripts
    agent-acompact-{hash}.jsonl              # Compaction subagent transcripts
```

Related directories:

- `~/.claude/file-history/{sessionId}/` — File version backups
- `~/.claude/todos/{sessionId}-agent-{id}.json` — Task lists
- `~/.claude/debug/{sessionId}.txt` — Debug logs

5.2 Universal Fields

These fields appear on most JSONL lines:

Field	Type	Description
`type`	string	Line type discriminator

`uuid`	UUID	Unique identifier for this line
`parentUuid`	UUID | null	Previous line in conversation chain
`sessionId`	UUID	Session identifier
`timestamp`	ISO 8601	When this line was written
`isSidechain`	boolean	`true` for subagent messages
`cwd`	string	Current working directory
`version`	string	Claude Code version (e.g., `"2.1.47"`)
`gitBranch`	string	Current git branch
`userType`	string	Always `"external"`
`slug`	string	Whimsical session name (e.g., `"memoized-jumping-hinton"`)
`teamName`	string	Present when using agent teams

5.3 Line Type: `user`

User prompts, tool results, compaction summaries, skill injections.

Additional fields:

Field	Type	Description
`message`	object	`{ role: "user", content: string | ContentBlock[] }`
`isMeta`	boolean	Hidden from user UI (system prompts, skill injections)
`isCompactSummary`	boolean	Compaction summary injected after compact boundary
`isVisibleInTranscriptOnly`	boolean	Stored in file but not sent to API
`thinkingMetadata`	object	`{ maxThinkingTokens: number }`
`permissionMode`	string	e.g., `"bypassPermissions"`
`todos`	array	Task list state at time of message
`planContent`	string	Plan text when in plan mode
`toolUseResult`	object	Metadata about a tool result (skill/tool success info)
`sourceToolAssistantUUID`	UUID	Links tool result back to the assistant message
`sourceToolUseID`	string	Links to the specific tool_use block
`agentId`	string	Short hash identifying a subagent (subagent transcripts only)

When `content` is a string: plain user prompt, XML-wrapped commands (`<command-name>`), compact summaries.

When `content` is an array: `tool_result` blocks returned to the model.

5.4 Line Type: `assistant`

Model responses.

Additional fields:

Field	Type	Description
`message`	object	Full Anthropic Messages API response (see below)
`requestId`	string	API request ID (e.g., `"req_011CYHgCeQ2EUm3xoZNVy6Wu"`)
`isApiErrorMessage`	boolean	Synthetic error message (auth failure, rate limit)
`error`	string	Error code when `isApiErrorMessage` is true
`agentId`	string	Subagent identifier (subagent transcripts only)

`message` object structure:

```
{
  "id": "msg_01HJ9vbRWfsva7Wo2sx7XkFK",
  "type": "message",
  "role": "assistant",
  "model": "claude-opus-4-6",
  "content": [ /* ContentBlock[] */ ],
  "stop_reason": "end_turn" | "tool_use" | null,
  "stop_sequence": null,
  "usage": {
    "input_tokens": 3,
    "output_tokens": 310,
    "cache_creation_input_tokens": 37910,
    "cache_read_input_tokens": 11029,
    "cache_creation": {
      "ephemeral_5m_input_tokens": 0,
      "ephemeral_1h_input_tokens": 37910
    },
    "service_tier": "standard",
    "inference_geo": "not_available",
    "server_tool_use": { "web_search_requests": 0, "web_fetch_requests": 0 },
    "iterations": [],
    "speed": "standard"
  }
}
```

Multiple JSONL lines can share the same `message.id` — CC streams content blocks as separate lines. The first line might have `text`, the second `thinking`, the third `tool_use`. All share one `message.id` and one `requestId`.

5.5 Content Block Types

Four types, inside `message.content` arrays:

`text`

```
{ "type": "text", "text": "Response text here..." }
```

`thinking`

```
{
  "type": "thinking",
  "thinking": "Let me analyze the user's request...",
  "signature": "EtcJCkYICxgCKkAPZJ5iTj0UE7oZJML..."
}
```

The `signature` is a cryptographic verification signature for extended thinking.

`tool_use`

```
{
  "type": "tool_use",
  "id": "toolu_016aAY5n6tgxdvfBLEBp6c4o",
  "name": "Skill",
  "input": { "skill": "superpowers:subagent-driven-development" },
  "caller": { "type": "direct" }
}
```

Known tool names: `Bash`, `Edit`, `Read`, `Write`, `Glob`, `Grep`, `Skill`, `Task`, `TaskCreate`, `TaskList`, `TaskUpdate`, `TaskOutput`, `EnterPlanMode`, `ExitPlanMode`, `WebSearch`, `WebFetch`, `NotebookEdit`, `SendMessage`, `TeamCreate`, `AskUserQuestion`, etc.

`tool_result`

```
{
  "type": "tool_result",
  "tool_use_id": "toolu_016aAY5n6tgxdvfBLEBp6c4o",
  "content": "output text...",
  "is_error": false
}
```

Always appears inside a `user` line's `message.content` array. The `tool_use_id` links back to the `tool_use` block in a preceding `assistant` line.

5.6 Line Type: `system`

System messages with a `subtype` discriminator:

Subtype	Description	Key Fields
`compact_boundary`	Compaction checkpoint	`compactMetadata.trigger` ("auto"/"manual"), `compactMetadata.preTokens`, `logicalParentUuid`

`local_command`	Slash command executed	`content` (XML-wrapped command)
`stop_hook_summary`	Hook execution results	`hookCount`, `hookInfos`, `hookErrors`, `preventedContinuation`
`turn_duration`	Agent turn timing	`durationMs`

5.7 Line Type: `progress`

Real-time progress updates with `data.type` discriminator:

`data.type`	Description	Notable Content
`hook_progress`	Hook execution streaming	`hookEvent`, `hookName`, `command`
`bash_progress`	Live stdout from Bash tool	`output`, `fullOutput`
`agent_progress`	Streaming subagent responses	Contains full nested assistant `message` object
`waiting_for_task`	Main agent blocked on subagent	`taskDescription`, `taskType`

5.8 Line Type: `file-history-snapshot`

File modification tracking for undo/restore:

```
{
  "type": "file-history-snapshot",
  "messageId": "7e976302-...",
  "isSnapshotUpdate": false,
  "snapshot": {
    "messageId": "7e976302-...",
    "timestamp": "2026-02-19T15:37:19.847Z",
    "trackedFileBackups": {
      "packages/cli/src/commands/hooks.ts": {
        "backupFileName": "2b82df92d54ee5b7@v1",
        "version": 1,
        "backupTime": "2026-02-22T20:43:59.763Z"
      }
    }
  }
}
```

Backup files stored at `~/.claude/file-history/{sessionId}/{hash}@v{version}`.

5.9 Line Type: `queue-operation`

Prompt queue management:

```
{
  "type": "queue-operation",
  "operation": "enqueue" | "dequeue" | "popAll" | "remove",
  "sessionId": "...",
```

```
  "content": "{ \"task_id\": \"bc6614a\", ... }"
}
```

5.10 Line Type: `summary`

Session title/summary (found at end of transcripts or in session index files):

```
{
  "type": "summary",
  "summary": "Linear Task Creation with Dependency Structure",
  "leafUuid": "4e9aaab5-..."
}
```

5.11 Compaction

Compaction is a two-line sequence:

1. **`system/compact_boundary`** — marks where old history ends. `parentUuid: null` (starts new chain), `logicalParentUuid` references the last pre-compaction message.
2. **`user` with `isCompactSummary: true`** — immediately follows. Contains a detailed summary of everything before the boundary. Has `isVisibleInTranscriptOnly: true` (stored in transcript but not resent to API).

The JSONL file grows (old lines preserved), but API calls after compaction only send summary + new messages. Auto-triggers at ~98% context usage.

5.12 Subagent Transcripts

When the `Task` tool is used:

1. Main transcript: `assistant` line with `tool_use` of `name: "Task"`
2. Main transcript: `progress` lines with `data.type: "agent_progress"` (streams subagent activity)
3. Separate file: `{sessionId}/subagents/agent-{agentId}.jsonl` — complete independent transcript
4. Main transcript: `user` line with `tool_result` when subagent finishes

Subagent transcripts have identical structure to main transcripts, with `isSidechain: true`, `agentId`, and `slug` on every line.

5.13 Message Linking

Messages form a chain via `uuid` + `parentUuid`:

- First user message: `parentUuid: null`
- Each subsequent message: `parentUuid` = predecessor's `uuid`
- Compaction: `compact_boundary` has `parentUuid: null` but `logicalParentUuid` references the last pre-compaction message
- Compact summary: `parentUuid` = compact boundary's `uuid`

---

6. Gap Analysis: What We Capture vs. What Exists

6.1 Line Types: 4 of 7 Processed

CC Line Type	Our Handling	Impact
`user`	**Parsed** into messages + content blocks	-
`assistant`	**Parsed**, grouped by `message.id`	-
`system`	**Parsed** as generic message (no subtype extraction)	Medium — compaction boundaries, turn durations invisible
`summary`	**Parsed** as standalone message	-
`progress`	**Skipped entirely** (in `SKIP_TYPES`)	High — agent_progress has full subagent streaming data
`file-history-snapshot`	**Skipped entirely**	Low — undo tracking, not conversation data
`queue-operation`	**Skipped entirely**	Low — internal bookkeeping

6.2 Fields Present in CC but Missing from `RawTranscriptLine`

Field	What It Contains	Impact
`slug`	Whimsical session name (e.g., `"memoized-jumping-hinton"`)	**Medium** — human-friendly session identifier
`isMeta`	Whether message is hidden from user UI	**High** — system prompts and skill injections counted as real messages
`isCompactSummary`	Marks compaction summary messages	**High** — related to compaction handling we don't do
`isVisibleInTranscriptOnly`	Present on compact summaries	Medium
`agentId`	Subagent identifier hash	**High** — can't reconstruct subagent conversations
`sourceToolAssistantUUID`	Links tool result → assistant message	**High** — enables tool_use→tool_result correlation
`sourceToolUseID`	Specific tool_use block this result responds to	**High** — same
`requestId`	API request ID	Medium — debugging
`isApiErrorMessage`	Marks synthetic error messages	**Medium** — error messages counted as real responses
`error`	Error code for API failures	Medium

`logicalParentUuid`	Pre-compaction parent reference	Medium
`teamName`	Agent team name	Low
`permissionMode`	e.g., `"bypassPermissions"`	Low
`todos`	Task list state	Medium — could reconstruct task flow
`planContent`	Plan text when in plan mode	Medium — plan analysis
`thinkingMetadata`	`{ maxThinkingTokens }`	Low
`toolUseResult`	Skill/tool result metadata	Low

6.3 System Subtypes: Not Differentiated

All `system` lines become `message_type: "system"` with no subtype extraction:

Subtype	Lost Data
`compact_boundary`	`compactMetadata.trigger`, `compactMetadata.preTokens`, `logicalParentUuid`
`turn_duration`	`durationMs` (per-turn timing)
`stop_hook_summary`	Hook execution results
`local_command`	Which slash commands were run

6.4 Usage Object: 4 of 10+ Fields Captured

Field	Captured?	Notes
`input_tokens`	Yes	
`output_tokens`	Yes	
`cache_creation_input_tokens`	Yes	
`cache_read_input_tokens`	Yes	
`cache_creation.ephemeral_5m_input_tokens`	**No**	5-minute ephemeral cache
`cache_creation.ephemeral_1h_input_tokens`	**No**	1-hour ephemeral cache
`service_tier`	**No**	"standard" etc.
`inference_geo`	**No**	Geographic routing
`server_tool_use.web_search_requests`	**No**	Web search count
`server_tool_use.web_fetch_requests`	**No**	Web fetch count
`iterations`	**No**	Internal iteration count
`speed`	**No**	Speed tier

6.5 Content Block Fields: Minor Gaps

Field	Captured?
`thinking.signature`	**No** (cryptographic verification)
`tool_use.caller`	**No** (`{ type: "direct" }`)

6.6 Structural Gaps

Compaction: Schema Ready, Logic Missing

The DB has `compact_sequence` and `is_compacted` columns. The parser hardcodes them to `0` and `false`. No code reads `isCompactSummary`, `compact_boundary`, or increments the sequence. **Can't distinguish pre-compaction from post-compaction messages.**

UUID Chain: Discarded

CC's transcript forms a linked list via `uuid` + `parentUuid`. We generate new ULIDs for `TranscriptMessage.id` and discard both. The `raw_message` JSONB preserves `message` content but `uuid` / `parentUuid` are **line-level fields**, not inside `message`. **Conversation tree structure is lost.**

Subagents: Not Handled

CC stores subagent transcripts as separate `.jsonl` files at `{sessionId}/subagents/`. Our system:

- Only uploads/parses the **main** session JSONL
- Counts subagents by `tool_name === "Task"` in content blocks
- Has no mechanism to discover, upload, or parse subagent transcripts
- **Cannot reconstruct what any subagent actually did**

Cost Model: Hardcoded Single-Model Pricing

```
const PRICE_INPUT_PER_MTOK = 3.0;      // Sonnet pricing
const PRICE_OUTPUT_PER_MTOK = 15.0;
const PRICE_CACHE_READ_PER_MTOK = 0.3;
const PRICE_CACHE_WRITE_PER_MTOK = 3.75;
```

Does not account for different models (Opus and Haiku have very different rates), despite the `model` field being available per-message.

`isMeta` Filtering: Missing

Meta messages (`isMeta: true`) are system prompts, skill injections, and command caveats. We count these as regular messages, inflating `user_messages` and `total_messages` stats.

---

7. Real Transcript Snippets

All snippets from real Claude Code transcripts on this machine, with paths for reference.

7.1 Main Session — First Lines (hook progress + file snapshot + user message)

Source: `~/.claude/projects/-Users-johnmemon-Desktop-fuel-code/008d3304-e9ed-4ed4-b16c-66801bbaf6b7.jsonl` (3.6MB, 926 lines)

Line 1 — `progress` (hook firing):


```
{
  "type": "progress",
  "parentUuid": null,
  "isSidechain": false,
  "userType": "external",
  "cwd": "/Users/johnmemon/Desktop/fuel-code",
  "sessionId": "008d3304-e9ed-4ed4-b16c-66801bbaf6b7",
  "version": "2.1.47",
  "gitBranch": "main",
  "data": {
    "type": "hook_progress",
    "hookEvent": "SessionStart",
    "hookName": "SessionStart:startup",
    "command": "bash -c '#!/bin/bash\n# Fuel Hook — Auto-generated\n...'"
  },
  "parentToolUseID": "561f6bda-8ad4-405f-9013-766a5f42a266",
  "toolUseID": "561f6bda-8ad4-405f-9013-766a5f42a266",
  "timestamp": "2026-02-19T15:36:49.762Z",
  "uuid": "aa61513a-d47d-4daa-b491-2fb16f20e207"
}
```


Line 3 — `file-history-snapshot`:


```
{
  "type": "file-history-snapshot",
  "messageId": "7e976302-5b18-4479-b786-938f20356fb2",
  "snapshot": {
    "messageId": "7e976302-5b18-4479-b786-938f20356fb2",
    "trackedFileBackups": {},
    "timestamp": "2026-02-19T15:37:19.847Z"
  },
  "isSnapshotUpdate": false
}
```


Line 4 — `user` with `isMeta: true` (system injection):


```
{
  "type": "user",
  "parentUuid": "14f1618b-e6e6-4a9a-bfc0-d0b04499ff7d",
  "isSidechain": false,
  "isMeta": true,
  "message": {
    "role": "user",
    "content": "<local-command-caveat>Caveat: The messages below were generated by
the user while running local commands. DO NOT respond to these messages or otherwise
consider them in your response unless the user explicitly asks you to.</local-
command-caveat>"
  },
```

```
  "uuid": "282b1137-a90a-45e2-9f6b-e1d9e0ab1f51",
  "timestamp": "2026-02-19T15:37:19.846Z"
}
```


Line 8 — Real user prompt (with `todos` and `permissionMode`):


```
{
  "type": "user",
  "parentUuid": "b6c51a7b-969c-434a-bd9c-01bce22f9223",
  "message": {
    "role": "user",
    "content": "Create an agent team to implement phase 2 @tasks/phase-2/
@tasks/phase-2/dag.md "
  },
  "uuid": "9ea7b0b3-457c-431f-8dfe-12fce61b4fb7",
  "timestamp": "2026-02-19T15:37:45.883Z",
  "todos": [],
  "permissionMode": "bypassPermissions"
}
```


7.2 Assistant Message — Streaming Multi-Line (text + thinking + tool_use)

Source: Same file, lines 9-11

Three consecutive JSONL lines share `message.id: "msg_01HJ9vbRWfsva7Wo2sx7XkFK"`:

Line 9 — text block (empty newlines):


```
{
  "type": "assistant",
  "message": {
    "model": "claude-opus-4-6",
    "id": "msg_01HJ9vbRWfsva7Wo2sx7XkFK",
    "content": [{ "type": "text", "text": "\n\n" }],
    "stop_reason": null,
    "usage": {
      "input_tokens": 3,
      "cache_creation_input_tokens": 37910,
      "cache_read_input_tokens": 11029,
      "cache_creation": {
        "ephemeral_5m_input_tokens": 0,
        "ephemeral_1h_input_tokens": 37910
      },
      "output_tokens": 11,
      "service_tier": "standard"
    }
  },
  "requestId": "req_011CYHgCeQ2EUm3xoZNVy6Wu"
}
```


Line 10 — thinking block (same `message.id`):

```
{
  "type": "assistant",
  "message": {
    "id": "msg_01HJ9vbRWfsva7Wo2sx7XkFK",
    "content": [{
      "type": "thinking",
      "thinking": "The user wants me to create an agent team to implement Phase
2...",
      "signature": "EtcJCkYICxgCKkAPZJ5iTj0UE7oZJML..."
    }]
  }
}
```


Line 11 — tool_use block (same `message.id`):


```
{
  "type": "assistant",
  "message": {
    "id": "msg_01HJ9vbRWfsva7Wo2sx7XkFK",
    "content": [{
      "type": "tool_use",
      "id": "toolu_016aAY5n6tgxdvfBLEBp6c4o",
      "name": "Skill",
      "input": { "skill": "superpowers:subagent-driven-development" },
      "caller": { "type": "direct" }
    }],
    "usage": {
      "output_tokens": 310,
      "server_tool_use": { "web_search_requests": 0, "web_fetch_requests": 0 },
      "speed": "standard"
    }
  }
}
```


Our parser correctly merges these 3 lines into **one** `TranscriptMessage` + 3 `ParsedContentBlock` rows.

7.3 Compact Boundary + Compact Summary

Source: Same file, lines 447-448

Line 447 — `system/compact_boundary`:


```
{
  "type": "system",
  "subtype": "compact_boundary",
  "content": "Conversation compacted",
  "parentUuid": null,
  "logicalParentUuid": "d6e51577-2513-4d91-b3dd-ee8f418883f2",
  "teamName": "phase-2-impl",
  "slug": "memoized-jumping-hinton",
  "isMeta": false,
```

```
  "timestamp": "2026-02-19T16:12:02.906Z",
  "uuid": "4020e017-c550-4440-a475-b58676df8a80",
  "level": "info",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 168396
  }
}
```


Line 448 — `user` with `isCompactSummary: true`:


```
{
  "type": "user",
  "parentUuid": "4020e017-c550-4440-a475-b58676df8a80",
  "isVisibleInTranscriptOnly": true,
  "isCompactSummary": true,
  "message": {
    "role": "user",
    "content": "This session is being continued from a previous conversation that
ran out of context. The summary below covers the earlier portion of the
conversation.\n\nAnalysis:\nLet me chronologically analyze the conversation:\n\n1.
User requested creating an agent team to implement Phase 2...\n[...detailed summary
of all prior work, files, errors, pending tasks...]\n\nIf you need specific details
from before compaction, read the full transcript at:
/Users/johnmemon/.claude/projects/-Users-johnmemon-Desktop-fuel-code/008d3304-e9ed-
4ed4-b16c-66801bbaf6b7.jsonl"
  },
  "uuid": "67c53da6-f8e4-479d-ade2-354bbd4552fb",
  "timestamp": "2026-02-19T16:12:02.906Z"
}
```


What our parser does with this: Line 447 becomes a generic `message_type: "system"` row with no subtype metadata. Line 448 becomes a normal `message_type: "user"` row. `compact_sequence` stays 0, `is_compacted` stays false on all rows.

7.4 Turn Duration

Source: Same file, line 196


```
{
  "type": "system",
  "subtype": "turn_duration",
  "durationMs": 182545,
  "parentUuid": "297d18b7-28fe-4534-8a23-8a97f345b7fd",
  "teamName": "phase-2-impl",
  "slug": "memoized-jumping-hinton",
  "timestamp": "2026-02-19T15:40:48.430Z",
  "uuid": "c56de8d1-d75d-4f35-8964-aa25edd28979",
  "isMeta": false
}
```

What our parser does with this: Becomes a `message_type: "system"` row. The `durationMs: 182545` (3 minutes) is available in `raw_message` JSONB but not extracted into any column.

7.5 Subagent Transcript

Source: `~/.claude/projects/-Users-johnmemon-Desktop-fuel-code/0f6d37a0-49bf-41dc-833c-01d0a435b57a/subagents/agent-a6fe488.jsonl` (5.4KB, 5 lines)

Line 1 — User message with teammate coordination:


```
{
  "type": "user",
  "parentUuid": "a2b4fa1b-d0ec-4b89-bbdc-24b589210f83",
  "isSidechain": true,
  "agentId": "a6fe488",
  "slug": "transient-herding-neumann",
  "message": {
    "role": "user",
    "content": "<teammate-message teammate_id=\"phase-7-reviewer\"
color=\"purple\">\n{\"type\":\"task_assignment\",\"taskId\":\"4\",\"subject\":\"Phase
7 downstream impact review\",...}\n</teammate-message>"
  }
}
```


Line 2 — Assistant with full usage including cache breakdown:


```
{
  "type": "assistant",
  "isSidechain": true,
  "agentId": "a6fe488",
  "slug": "transient-herding-neumann",
  "message": {
    "model": "claude-opus-4-6",
    "id": "msg_01P98hn8Bb8aGZAKRM95dbGM",
    "content": [{ "type": "text", "text": "This task assignment is a duplicate..."
}],
    "usage": {
      "input_tokens": 3,
      "cache_creation_input_tokens": 248,
      "cache_read_input_tokens": 119929,
      "cache_creation": {
        "ephemeral_5m_input_tokens": 248,
        "ephemeral_1h_input_tokens": 0
      },
      "output_tokens": 1,
      "service_tier": "standard"
    }
  },
  "requestId": "req_011CYJBHcddGxsTU3vsBowZd"
}
```

Line 4 — Tool result with `sourceToolAssistantUUID` linking:


```
{
  "type": "user",
  "isSidechain": true,
  "agentId": "a6fe488",
  "message": {
    "role": "user",
    "content": [{
      "tool_use_id": "toolu_01BHoD8nUJ7qps7nfkYbwFME",
      "type": "tool_result",
      "content": "#1 [in_progress] Phase 4 downstream impact review (phase-4-
reviewer)\n#2 [in_progress] Phase 5 downstream impact review ..."
    }]
  },
  "toolUseResult": {
    "tasks": [
      { "id": "1", "subject": "Phase 4 downstream impact review", "status":
"in_progress", "owner": "phase-4-reviewer" },
      { "id": "4", "subject": "Phase 7 downstream impact review", "status":
"completed", "owner": "phase-7-reviewer" }
    ]
  },
  "sourceToolAssistantUUID": "a907e1e4-4c58-4ff3-b2fd-5fcc9eb25a15"
}
```


What our parser does with this: Nothing — this file is never uploaded, discovered, or parsed.

7.6 Compact Subagent Transcript

Source: `~/.claude/projects/-Users-johnmemon-Desktop-fuel-code/3f568070-2f58-4b26-b035-dfaecb935d9a/subagents/agent-acompact-ae3a23.jsonl` (5.5KB, 1 line)

A special subagent spawned to generate the compaction summary. Filename pattern `agent-acompact-*` distinguishes these from regular subagents. Contains a massive summarization prompt with instructions to produce a `<summary>...</summary>` block.

---

8. References

Official Documentation

- How Claude Code Works — Session storage, compaction behavior, resume/fork mechanics
- Claude Code Statusline — Full JSON schema for session data including context window usage
- Claude Code Hooks — Hook inputs include `session_id`, `transcript_path`

Community Research

- Claude Code Data Structures (samkeen gist) — Most comprehensive reverse-engineered field reference
- claude-code-log (daaain) — Python parser handling all message types, token tracking, HTML output
- claude-code-transcripts (simonw) — JSONL→HTML converter with gist sharing
- claude-JSONL-browser (withLinda) — Web-based JSONL→Markdown viewer

- ClaudeCodeJSONLParser (amac0) — Parser with git commit timeline integration

Analysis & Articles

- Analyzing Claude Code Logs with DuckDB (liambx) — DuckDB-based analysis of transcript fields, tool usage, error patterns
- What Happens When You /compact (dev.to/rigby_) — Detailed compact boundary analysis with measured token reduction (87%)
- Calculate Claude Code Context Usage (codelynx) — Token calculation, sidechain filtering, auto-compact threshold (~98%)
- Context Window & Compaction (DeepWiki) — Session management architecture

GitHub Issues

- #20531 TaskOutput returns full JSONL transcript — Subagent transcript structure and isSidechain behavior
- #16944 Document subagent auto-compaction — Subagent compaction boundary format with `compactMetadata`
- #26771 Indexed transcript references in compaction — Feature request for surgical context recovery from raw transcripts

Transcript Retention

Transcripts are auto-deleted after 30 days by default. To prevent: add `"cleanupPeriodDays": 99999` to `~/.claude/settings.json`. (source)