Names Are Structure: A One-Key Collision at the MCP Result Boundary

Technical Incident Report: Defect D-20, AtlasClose Chat Door

Field	Value
System	AtlasClose Chat (single live record, Oracle Cloud deployment)
Component	Chat door (the door program, served over MCP HTTP), tool `chat_read_file`
Defect ID	D-20, defect register, AtlasClose operations
Failure reported	2026-11-20, by the affected party (Gippy, ChatGPT, external audit)
Correction deployed	2026-11-20, door v4
Acceptance	2026-11-20, by the affected party in its own runtime (Section 7.2)
Status	CLOSED: fixed and accepted
Door version before	v3, sha256 `777141db...`
Door version after	v4, sha256 `d9a6500b...`
Classification	Reserved-name collision between a payload key and the protocol envelope (Section 8.1)
Report type	Technical incident report: evidence, method, root cause, correction, acceptance, discussion

Epistemic labels. This report uses three labels throughout, and every load-bearing claim carries one. **KNOWN** means verified by a named check whose output is cited. **INFERRED** means the evidence points there but the mechanism itself cannot be observed from the systems available to the investigators. **UNKNOWN** means not established by anything examined. The distinction is not decoration: in this incident the difference between the second and first label is the difference between a cure that is proven and a mechanism that is not, and the report keeps them separate at every step.

Abstract

An external AI client (ChatGPT, operating in AtlasClose Chat as the bot Gippy) could invoke every tool of the AtlasClose chat door except one. For `chat_read_file` alone, the client's runtime surfaced an empty object `{}` on every call: no payload, no error, no partial data. The server's own record showed success on every one of those calls; the door's log recorded the tool executing with its error flag false, and a live wire capture confirmed a well-formed payload leaving the server. Diagnosis proceeded by hypothesis elimination. Size was eliminated experimentally (the failure is invariant from 9,341 characters down to 100). Transport, envelope framing, and tool declaration were eliminated by construction (all tools share one wrapper, one transport, one schema builder). Structural comparison of every tool's result object then isolated a single singularity: `chat_read_file` was the only tool whose result carried a top-level key named `content`, which is also the name of the enclosing MCP result envelope's own field. The correction renamed that one key to `record`, in that one tool, and changed nothing else. The affected party then re-ran the read in its own run-

time and retrieved the complete document, 9,341 of 9,341 characters in 4 of 4 slices, which closed the defect. The client-side mechanism that discarded the payload remains INFERRED and has never been confirmed by the client vendor; the singularity, the cure, and the acceptance are KNOWN. The same trigger and the same remedy class were independently documented by human engineers in a public OpenAI Developer Community thread covering three unrelated MCP integrations (Gitea, Nextcloud, Linear), where the vendor's root-cause determination was still outstanding at last report. This report presents the evidence, the method, the correction and its process controls, and a discussion of why a single key name is load-bearing at a protocol boundary, what it means to engineer at this level of detail, and what the AI participants in this incident did and did not change about how such a defect is found.

1. Introduction

1.1 What happened, in one paragraph

AtlasClose Chat is a multi-party chat system whose record is a hash-chained, author-stamped ledger, and whose participants include AI systems connected from outside (Section 2). On 2026-11-20 an outside AI, Gippy (ChatGPT), completed an audit of its own ability to work in the system and reported a precise failure: it could communicate, search, and send files, but it could not retrieve the complete contents of a document held in the chat. Every call to the file-read tool returned an empty object. The defect was assigned ID D-20, moved to the top of the build queue by the system owner, diagnosed at the wire level, corrected by renaming a single JSON key, and accepted the same day by the affected party's own retest. This report is the technical record of that incident.

1.2 Why this incident merits a technical report

Three properties make the incident worth documenting at this standard rather than in a changelog line. First, the failure was silent and total on exactly one side of a protocol boundary: every server-side signal said success, and the payload was destroyed inside a closed client. Failures of this shape are systematically under-documented because the party who can see the symptom cannot see the cause, and the party who can see the cause (the vendor) has published nothing. Second, the corrective change is one word, which makes the incident a clean natural experiment: with every other variable held constant, a single identifier is shown to be the difference between a working and a non-working cross-system integration. Third, the failure report, the acceptance test, and part of the diagnosis were produced by AI systems acting as participants rather than as tools, which raises a question the field currently answers mostly with anecdote: what does an AI participant actually change in the lifecycle of a defect like this? Section 8.3 addresses that question against the public counter-evidence.

1.3 Scope and sources

This report covers defect D-20 only: the return path of `chat_read_file` results to one external client runtime. It does not cover the door's other tools except as controls in the comparison of Section 4.4, and it does not cover the client's other behaviors. Primary sources are the door source code at versions v3 and v4, the door's call log, a live wire capture taken through a second connector, the chat ledger of the main thread, the pre-change freeze manifest, the independent pre-deploy audit, and the affected party's own published test results in that thread. Public sources are cited in the

References and are used for classification, precedent, and the counterfactual analysis, never as evidence about this system's internals.

2. System context

2.1 The door

The chat door is the MCP server through which outside AIs reach the chat. It is a single Python module, served over HTTP on the deployment host. A connecting client authenticates with a per-bot token; the door resolves the token to a bot identity and a role, and dispatches tool calls under a global lock, setting the caller's identity (bot, route, and role) per request. Every call is logged to a per-call log file with timestamp, bot, tool, and an error flag; the log records names only, never contents. The board (the human's screen) is a separate program and was not modified by this incident's correction.

Two properties of the door matter for what follows. First, all tool results return through one shared wrapper in `handle()`: the handler's result object is serialized with `json.dumps(result, indent=2)` into a single text content block, with the MCP `isError` flag alongside, over plain-JSON HTTP with `Content-Length` set. There is one envelope construction for every tool; no tool has its own framing. Second, the door's `tools/list` declares only `name`, `description`, and `inputSchema` for each tool. No tool declares an `outputSchema`, and `chat_read_file`'s `inputSchema` is built by the same helper as every other tool's.

2.2 The MCP result envelope

Under the Model Context Protocol, a tool call returns a `CallToolResult`:

```
{
  "_meta": {},
  "content": [ { "type": "text", "text": "<result>" } ],
  "structuredContent": {},
  "isError": false
}
```

The `content` array is required and carries the unstructured result as typed content blocks; `structuredContent` is optional and carries a structured JSON object; `_meta` and `isError` complete the envelope [1]. In this door's implementation the result object produced by a tool handler is itself JSON, carried as the string value of the single text block's `text` field. The wire shape for a file read before the correction was therefore:

```
{
  "content": [
    { "type": "text",
      "text": "{\"thread_id\": \"...\", \"entry\": 250, \"filename\": \"...\", \"sha256\":
\"...\", \"content\": \"<document text>\"}" }
  ],
  "isError": false
}
```

Note the two occurrences of the name `content` at two different levels of interpretation: the envelope's field, which the client parses as protocol, and a key inside the JSON string, which is payload data unless the client chooses to parse and interpret that string. On the wire there is exactly one `content` key; the second exists only after the inner JSON is parsed. This distinction is load-bearing in Section 5.3.

2.3 Tool result shapes

The door exposes thread tools (read, reply, read another chat, search, send file, read file, briefing) and, for auditor-role callers, a separate audit tool set. The result objects of the thread tools, as constructed by their handlers at door v3, are:

Tool	Top-level keys of result object	Carries document text under
`chat_read`	thread fields with `messages` and `text` nested	nested `text`
`chat_read_chat`	`record`	`record`
`chat_search`	`results`	`results`
`chat_send_file`	`entry`, `sha256`	(returns a receipt, no text)
`chat_briefing`	named section keys	named sections
`chat_read_file`	`thread_id`, `entry`, `filename`, `sha256`, `ext`, `size`, `content`	`content`

Table 1: Result shapes at door v3. `chat_read_file` is the only tool whose result object has a top-level `content` key; it is also the only tool that failed in the affected client. Section 4.4 develops this comparison.

`chat_read_file` supports whole-file reads and sliced reads: with `start` and `count` arguments it returns a character slice plus `offset`, `returned_chars`, `total_chars`, and a `more` flag. Results are capped at 256 KB. For the affected party, access was pinned by the system owner to a single document (thread entries 250 and 260); any other file returns a `NOT_SCOPED` refusal. The pin, the membership rules, and every error path are part of the control surface the correction was forbidden to disturb.

2.4 The document under test

The document is the owner's Post 3 article, "The Answer to AI Is More AI, Not Less," filed in the main thread at entries 250 and 260. It is 9,341 characters, sha256 `286656c7...`. The article itself is immaterial to the defect; it served as a fixed, hash-identified test object whose complete retrieval could be verified character for character and by hash on both

sides of the boundary. A document whose integrity is independently checkable is what makes the acceptance test of Section 7.2 a measurement rather than an impression.

2.5 The parties

- **Gippy**: ChatGPT, connected to the door from its own runtime as an outside AI. The affected party: the failure existed only inside its connector's handling of results. Also the reporting party: the defect entered the record through Gippy's own audit.
- **Muse**: a second outside AI, connected through the same door with its own token. Its connector reads the text block literally and was never affected. It served as the control client for the wire capture and post-deploy verification.
- **The operator side**: the door, board, dispatcher, and ledger on the deployment host, plus the human owner, who set priorities, scoped the affected party's access, and confirmed acceptance at the point of use.
- **The independent auditor**: a separate AI instance that reviewed the correction before deploy and did not write it (Section 6.3).

3. The failure as reported by the affected party

3.1 The audit that surfaced it

Gippy's end-to-end audit evaluated whether an outside AI can perform ordinary document work in the system: receive an instruction, locate a document, retrieve its complete contents without losing data, analyze it, file findings, and verify availability. Its published conclusion, entered in the main thread and relayed to the owner: basic external-agent communication passes; document-based work fails, blocked at step 3 (retrieval); overall production readiness unverified. Its recommendation was that the next milestone should be one ordinary task completed end to end by an outside AI, not another architectural feature. The send direction was proven separately: a 61-byte test fixture Gippy transmitted landed intact in the thread (its entry 150). The defect was therefore specific to the read-back direction of one tool. Impact was bounded and is stated exactly: no data was lost, altered, or exposed at any point, because the payloads existed complete on the server throughout; the harm was the total unavailability of one function, document retrieval, to one class of client, for as long as the defect stood, which blocked that client's document-based work and left the system's overall production readiness unverified in its audit.

3.2 Symptom specification

Across all of Gippy's tests (thread entries 287, 301, 302, 304), the symptom was invariant:

Variable	Values tested	Result in Gippy's runtime
Target	entry 250, entry 260, lookup by filename	`{}` every time
Mode	whole file, sliced	`{}` every time
Slice size	3,000, 500, 400 characters; 100 characters in wire review	`{}` every time
Payload size	9,341 characters down to 100 characters	`{}` every time

Variable	Values tested	Result in Gippy's runtime
Error surface	none: no error object, no message, no hash, no partial text	empty object only
Other tools, same session	read, read_chat, search, send_file, briefing	arrived intact

Table 2: Symptom specification for D-20, from the affected party's tests and the subsequent wire review.

On the server side, the same calls appear in the door's call log with the error flag false, and the payloads the door constructed were complete and well-formed (Section 4.3). The failure therefore had a property that defines the whole investigation: it was invisible from every vantage point except inside the affected client. No log line, on either side that the investigators could read, recorded anything wrong.

3.3 Timeline

When	Event
2026-11-19	Post 3 article filed in the chat (entries 250 and 260) and published to LinkedIn and X. Door at v3; during the v3 period, sliced reads had been added on the size hypothesis (H1, Section 4.2)
2026-11-20	Gippy's end-to-end audit concludes: document-based work blocked at step 3, retrieval. Failure entered on the record (thread entry 287)
2026-11-20	Gippy's retests at 3,000, 500, and 400 characters, with wire review at 100 characters (entries 301, 302, 304): `{}` at every size, falsifying H1
2026-11-20	Owner moves D-20 to the head of the build queue. Diagnosis by wire capture and structural comparison (Section 4). Pre-change freeze taken; independent audit returns PASS; door v4 deployed by hash-guarded install, door service only
2026-11-20	Technical fix report filed in the thread (entry 308); retest instructions posted (entry 310)
2026-11-20, about 13:12 PT	Acceptance: Gippy re-runs the read in its own ChatGPT runtime and retrieves the complete article (Section 7.2). Acceptance entered on the record (entry 312); the pre-change freeze recorded by the Secretary's office (entry 313); defect register updated to FIXED AND ACCEPTED

Clock times are given only where the record carries them; the sequence within 2026-11-20 is as logged in the thread and the fix report.

4. Method

4.1 Evidence discipline

The investigation followed the evidence discipline this report also uses: a claim is only as strong as the check that produced it, hypotheses are eliminated by named tests rather than by plausibility, and any proposition whose mechanism cannot be observed is labeled INFERRED and kept out of the root-cause statement. Three evidence sources were treated as authoritative, in order: the wire (what actually left the server, captured live), the source (the door code at the frozen v3 hash), and the affected party's own test results (what its runtime surfaced). The server's call log was treated as

corroboration, not proof, for a reason this incident makes vivid: the log records that a call executed and what the door constructed, and on every failing call it recorded success. A log that cannot see the failure cannot be the instrument that rules causes in or out.

4.2 Hypothesis elimination

Five hypotheses account for the plausible causes of an empty result at a client. Each was tested; the table records the test and the outcome.

ID	Hypothesis	Test	Outcome
H1	Payload too large; the client truncates or drops oversized results	Affected party retested at 3,000, 500, and 400 characters; wire review at 100 characters (thread entries 301, 302, 304)	ELIMINATED. `{}` at every size; a 40-fold reduction changed nothing
H2	Transport or framing fault (HTTP, chunking, content length)	Live wire capture through the control client: the `chat_read_file` slice response is a well-formed 903-byte payload for a 400-character slice; framing identical across tools	ELIMINATED. The payload left the server intact and well-formed
H3	Envelope construction differs for the failing tool	Source inspection: one shared wrapper in `handle()` builds every result; one transport for all tools	ELIMINATED by construction. No per-tool envelope exists
H4	Tool declaration fault (schema, missing output declaration)	`tools/list` comparison: only `name`, `description`, `inputSchema` declared for any tool; no `outputSchema` anywhere; `chat_read_file`'s `inputSchema` built by the shared helper	ELIMINATED. The declaration surface is uniform
H5	A structural property of the result object itself triggers client-side discarding	Structural comparison of every tool's result object against the pass/fail record (Section 4.4)	SUSTAINED as the discriminator. Exactly one structural difference exists, and it separates the failing tool from all working tools

Table 3: Hypothesis elimination for D-20. H1 deserves a note: it was the investigation's own first diagnosis, recorded in the one-record build report and the defect register during the door v3 period, when sliced reads were added on the theory that size was the cause. Gippy's sliced retests falsified it. The earlier diagnosis is superseded in the register, and it is retained here deliberately: an elimination table that hides the investigators' own wrong first hypothesis would misrepresent the method.

4.3 The wire capture

A live capture through the control client's connector (Muse, 2026-11-20) recorded the response to a sliced `chat_read_file` call (400 characters requested) as a single text content block of 903 bytes total, containing the complete JSON result object: thread and entry identifiers, filename, the document's sha256, and the 400 requested characters under the `content` key, with `isError` false. This capture is the investigation's ground truth for the server side: whatever happened to the payload happened after this point, inside the receiving client, on a payload that was complete, well-formed, and small.

4.4 Structural comparison

With H1 to H4 eliminated, the remaining space is the result object itself. Every tool's handler and every pass-through layer was inspected, and the result objects were compared against the pass/fail record in the affected client:

- `chat_read_chat` returns `{"record": string}`: passes.
- `chat_search` returns `{"results": string}`: passes.
- `chat_send_file` returns `{"entry", "sha256"}`: passes.
- `chat_read` returns thread fields with the text nested under `text`: passes.
- `chat_briefing` returns named section keys: passes.
- `chat_read_file` returns an object whose top level includes `"content"`: fails, at every size, in every mode.

Exactly one structural property separates the failing tool from the working set: a top-level key named `content` in the payload object, the same name as the MCP envelope's own field (Section 2.2). No other result object, at the top level or at sampled nested levels, uses that name; the only other occurrence in the system is the envelope itself. This is a KNOWN finding: it is a fact about the source, verified against every handler, not a theory about the client.

5. Root cause analysis

5.1 Root cause statement

Root cause (KNOWN). The `chat_read_file` result object carried a top-level payload key, `content`, that duplicates the name of the MCP result envelope's own `content` field. It was the only tool result in the system with this property. In the affected client, results carrying that property were surfaced to the model as an empty object, with no error. Renaming the payload key removed the property, and the same client then retrieved the same document completely. The defect is closed on this statement; it does not depend on the mechanism proposed in Section 5.2.

5.2 The mechanism, labeled INFERRED

The mechanism that best fits every observation is: the affected connector, on parsing the JSON inside the text block, treats an object carrying a top-level `content` field as itself a result envelope, finds no content blocks within it, and surfaces an empty object. This is INFERRED. It is consistent with all evidence: every tool without the key passes at all sizes; the one tool with the key fails at all sizes; the send-direction tool, whose result has no such key, passes; and a second client that reads the text block literally is entirely unaffected. But ChatGPT's internal result handling is closed and cannot be observed from any vantage point available to the investigators. No statement in this report asserts it as fact, and the client vendor has never confirmed it (Section 5.4). Alternative mechanisms that produce the same observable behavior (a normalization step that drops or shadows the inner object, a flattening collision in the envelope's namespace) cannot be distinguished from outside, and do not need to be: the corrective action is identical for all of them.

5.3 A precision point: where the collision lives

This incident is not a duplicate-key defect in the sense of RFC 8259 [14]. On the wire there is one `content` key, the envelope's; the payload travels inside a string, where its keys are data, not protocol. The collision can only occur at the moment a client parses the inner JSON and interprets it in a namespace where the envelope's vocabulary already has meaning. The correct classification is therefore a **reserved-name collision between a payload key and the protocol envelope**, equivalently describable as envelope-payload shadowing: a payload author reused a name the surrounding protocol had already claimed, and a consumer that interpreted the payload in the protocol's namespace resolved the ambiguity by discarding the payload. The established precedents for this class, and why mature protocols reserve names precisely to prevent it, are treated in Section 8.1.

5.4 Corroboration from the public record

The trigger and the remedy class are independently documented outside this system. An OpenAI Developer Community thread begun 2026-10-16, "ChatGPT MCP file reads return empty objects across Gitea and Nextcloud integrations," records the identical symptom across three unrelated MCP servers: Gitea `get_file_contents`, Nextcloud `nc_webdav_read_file`, and the official Linear integration's `get_document` and `fetch` all returned `{}` in ChatGPT while server-side tests over stdio and HTTP returned complete payloads with `isError` false, and a second client (Codex) read the same documents completely [2]. Testers in that thread eliminated size over a range from 20 to 149,907 characters. An update reporting tests of 2026-10-22 states the finding directly: ChatGPT shows `{}` when the tool result, a JSON text block, contains a field named `content`, and renaming that field fixes it; the reporter confirmed the rename (`content` to `text`) live on a third, independent server implementation, verified line by line. The same update states the epistemic position this report also takes: why the client drops the payload is not confirmed; the workaround is. As of the thread's 2026-10-28 update the failure remained reproducible against the official Linear connector, whose users cannot rename anything server-side, with a vendor support case open (case #17440919) and no vendor root-cause statement in the thread. No official vendor documentation of the trigger was found in this investigation's research. The public record therefore corroborates the singularity, the symptom, and the remedy, and is silent, as this report is, on the mechanism.

5.5 Contributing conditions

Two conditions on the server side did not cause the defect but shaped it. First, the door declares no `outputSchema` for any tool. The protocol permits text-only results, and the absence of a schema is not a violation; but a declared schema is the one channel through which a server can state its result shape in a form clients are directed to validate against, and its absence left the payload's shape entirely to client interpretation. Second, the payload key was chosen as a plain domain noun (`content` is the natural name for what a file-read returns) with no convention reserving envelope vocabulary. A circumstantial corroboration is worth noting: the client vendor's own MCP guide, in its worked example of a document-fetch tool, names the full-text field `text`, not `content`, with no reason stated [4]. That is consistency, not evidence, and it is recorded as such. Both conditions are addressed as engineering practice in Section 8.2; neither is assigned as cause, because the same two conditions hold for every other tool in the system, and only the key name separates failure from success.

6. The correction

6.1 The change

The correction is confined to the construction of `chat_read_file`'s result object in `t_read_file`. The document text now rides under the key `record` instead of `content`, in both the whole-file result and the sliced result. The key was not invented for this fix: `record` is the key `chat_read_chat` already returns, and that tool's results were arriving intact in the affected runtime, making `record` the one candidate key with direct empirical evidence of safe passage through the failing client. A comment at the change site states the reason and warns against renaming the key back.

```
// door v3, chat_read_file result object (payload level)
{ "thread_id": "...", "entry": 250, "filename": "...", "sha256": "...",
  "ext": "md", "size": 9341, "content": "<document text>" }

// door v4, chat_read_file result object (payload level)
{ "thread_id": "...", "entry": 250, "filename": "...", "sha256": "...",
  "ext": "md", "size": 9341, "record": "<document text>" }
```

	Door v3	Door v4
Key carrying the document text	`content`	`record`
Door code sha256	`777141db...`	`d9a6500b...`
Sliced result fields	`offset`, `returned_chars`, `total_chars`, `more`, `content`	identical, with `record` in place of `content`
Affected client surfaces	`{}`	the document text (accepted, Section 7.2)
All other tools	as before	byte-identical behavior; `tools/list` byte-identical

Table 4: The correction, before and after.

6.2 Containment

Nothing else changed, and the audit verified the negative space explicitly: the affected party's scope pin (one document readable, all others `NOT_SCOPED`), the membership bypass for the pinned bot, every error path, the slicing arithmetic, the 256 KB cap, all other tools and their result shapes, the `tools/list` output, the board program, and the ledgers. Containment matters technically, not just procedurally: the diagnosis indicted exactly one property of one object, so a correction that touched anything else would have destroyed the experiment's evidentiary value and risked converting a closed defect into an open regression. A fallback correction was designed and deliberately not built: returning the text in the exact shape of `chat_read_chat` (a single `{"record": string}` with metadata folded into header lines). It remains the prepared next step if the one-key correction had failed acceptance, and it was held back because it discards machine-readable fields and should ship only on evidence that the simpler correction is insufficient. Acceptance (Section 7.2) made it unnecessary.

6.3 Process controls

The correction shipped under the system's standing rules for work on the live record, which function here as experimental controls:

- **Freeze before change.** The full deployment tree was frozen before any edit: a freeze directory on the host, 4,404 files under a SHA-256 manifest (manifest sha256 `d06a4928...`), archived as a single archive file (363,458,560 bytes, sha256 `e88dc3dc...`). The frozen door module hashes to the v3 hash, so the freeze is a working rollback to the exact pre-change state.
- **Independent audit before deploy.** An auditor instance that did not write the correction reviewed it first and returned PASS with no blockers. The audit verified that the diff is confined to `t_read_file`'s result construction, that the `content` singularity holds against the source and every pass-through handler, that no local consumer reads the old key, that line endings and compilation are clean, and that the rollback path is sound. Its operational conditions were followed: freeze and backup hashes verified to v3 before install, installed file verified to the v4 hash after, and the single-file backup (kept on the host, hash verified) designated as the rollback path.
- **Hash-guarded deploy.** The install aborts unless the live file is exactly the expected v3 hash. Only the door service restarted. The board program was untouched (sha256 `6e0058ab...`), answered HTTP 200 after the restart, and all of the deployment's services were verified active. The one-record floor the owner had confirmed that morning, one live record serving phone and desktop, was intact after the change.

7. Verification and acceptance

7.1 Operator-side verification

Immediately after deploy, the corrected door was verified through the control client's connector against the live system:

- Whole read of entry 250 returns result keys `thread_id`, `entry`, `filename`, `sha256`, `ext`, `size`, `record`. No `content` key is present. The `record` value holds 9,341 characters and hashes to the document's sha256, `286656c7...`.
- Sliced reads at offsets 0, 3000, 6000, 9000 with count 3000 returned 3000, 3000, 3000, and 341 characters, with `more` reading true, true, true, false. The four slices tile the whole text exactly: no gaps, no overlaps.
- Entry 260 returns the identical text. A bad range still returns the `BAD_RANGE` error shape. `chat_threads` and `chat_read` are unaffected. `tools/list` is byte-identical to the pre-fix capture.

This verification establishes that the server now emits the corrected shape and that a literal-reading client receives the document. It does not, by itself, establish anything about the affected client, and under the acceptance standard of Section 7.3 it was not treated as closure.

7.2 Acceptance at the point of use

Acceptance was performed by the affected party, in its own runtime, against the retest posted in the main thread at entry 310 (the technical fix report was filed in the same thread at entry 308). Gippy re-ran the read in its ChatGPT runtime

and reported, in its own window: the complete Post 3 article retrieved directly from entry 250; 9,341 of 9,341 characters; 4 of 4 slices received; all offsets consecutive; the final slice marked complete; the server-reported SHA-256 consistent across all four slices. Its own summary line: the file-reading acceptance test passed, and this was the first successful completion of the external file-read operation that had been blocking document work. The owner observed and photographed the result in the affected client's window at about 13:12 PT on 2026-11-20, which is the acceptance evidence of record. The defect register was updated to FIXED AND ACCEPTED, and the acceptance and a reconciliation note were entered in the thread at entry 312.

One detail from the acceptance deserves record because it demonstrates the standard working as intended. Reading the retrieved document, Gippy flagged a discrepancy: the file's own header line states "NOT posted," while the article had in fact been published. The reconciliation, entered on the record, is that the header is a status line inside the document as filed; the article was posted to LinkedIn and X on 2026-11-19 and verified live at both; filed attachments are immutable, so the line was never updated. The affected party accepted the content on the strength of the hash and character counts, and declined, correctly, to treat the file alone as proof of publication. The file proves the content; the live posts prove the publication. An acceptance process that separates those two claims is doing exactly what it should.

7.3 Why acceptance sits at the point of use

The placement of acceptance is a technical decision, not a courtesy to the affected party, and this incident is a demonstration of why. The failure existed only in the composition of the whole path, server, wire, and one client's result handling; every component check available to the operator passed while the system was broken, and would have passed again after a wrong fix. This is the end-to-end argument in operational form: a function placed at the ends of a system cannot be verified from the middle, because the middle cannot see the failure the ends experience [16]. It is also the documented structure of this defect class in the public record: in the community thread, server-side reproduction over stdio and HTTP showed correct behavior throughout, and the failure was only ever observable in the client that consumed the result [2]. Accordingly, the register's closure rule for D-20 was fixed in advance: the defect closes only on the affected party's own successful read, in its own runtime, of the hash-identified document. Operator-side verification (Section 7.1) is a checkpoint on the way to that test, and was reported as such. A prepared fallback (Section 6.2) stood ready precisely because the acceptance test, not the investigators' confidence, was the decision point.

8. Discussion

8.1 Why one word is load-bearing at a layer boundary

To a human reader, a JSON key is a label: descriptive text attached to a value, chosen for clarity, changeable at will. At a protocol boundary that picture is wrong, and this incident is the proof by counterexample. A key in a machine-readable layer is not a label on the structure; it is part of the structure. The name `content` was already spoken for: it is the envelope's field, the word the protocol itself uses to say "the payload is here." When the payload used the same word for its own content, two meanings occupied one name in one namespace, and the consumer that interpreted the payload in the envelope's namespace resolved the collision by discarding the payload. Nothing about the document, the transport, or the tool's logic was wrong. The entire failure was a naming event at the exact joint between two systems.

Mature protocols treat this class as a first-order hazard and defend against it by reserving vocabulary, which is why the precedents are so uniform:

System	Reserved vocabulary	Consequence of collision	Defense adopted
System.Text.Json (.NET)	`$type`, `$id`, `$ref` for serializer metadata	Duplicate and ambiguous JSON, values that fail to round-trip	.NET 10 validates early and errors; guidance: avoid or rename conflicting names [9]
GraphQL	Any name beginning `__`	Payload names would collide with the introspection system	Prohibited by specification [10]
Protocol Buffers	Field numbers 19000 to 19999; names and numbers of deleted fields	Reused numbers decode silently as a different field: data corruption	`reserved` declarations; reuse treated as a defect class [12]
Google JSON APIs	Reserved property names list (`kind`, `id`, `items`, `error`, others)	A reserved name used with other semantics corrupts the contract	Style guide: never reuse reserved names; "there is no concept of JSON namespacing," resolve conflicts by choosing a new property name [11]
JSON (RFC 8259)	Object names SHOULD be unique	Mainstream parsers silently keep the last value: silent data loss, no error	A SHOULD that practice shows is load-bearing [14]

Table 5: Reserved-name precedents. The remedy in every mature case is the remedy applied here: the payload author renames.

Two features of the class deserve emphasis. The first is silence. In every close relative of this defect, the failure mode is not an error but an absence: a parser keeps the last duplicate without complaint, a client renders `{}` with `isError` nowhere in sight, a framework drops a channel without a warning. The .NET 10 change is notable precisely because it converts this class from silent ambiguity into a loud, early validation error; that is the direction the field is moving, and the direction this incident argues for. The second is that the ambiguity is structural, not incidental. The MCP specification itself leaves a neighboring gap: it defines `content` and `structuredContent` as the two result channels but states no precedence rule for clients when both are present, a gap formally recorded in SEP-1624 [3]. The vendor's own channel model for its apps surface assigns the channels different audiences (`structuredContent` for widget and model, `content` as optional narration, `_meta` hidden from the model) [5], which is a client-specific interpretation layered on the same underspecified contract. Observed client behavior diverges accordingly: one client drops all text blocks when `structuredContent` is present [7], another does not forward `content` to the model when a widget is rendered [6], a third has been reported dropping `structuredContent`-only results as "no result" while plain text survives [8]. A server therefore cannot control which channel a given client reads, and cannot assume that a shape that survives one client survives another. In a landscape like that, names and shapes are not cosmetics. They are the contract, and the contract is enforced, silently, by each consumer's private interpretation.

8.2 The level of detail as an engineering standard

The owner's design rule for this system, stated the same morning this defect closed, is that underneath every file and message there must be a machine-readable layer: the human surface and the machine layer are the same content in two forms, checkable against each other. D-20 is what that rule looks like when it is taken seriously as an engineering standard rather than as a slogan, because the incident was only diagnosable, correctable, and closable at the machine layer's level of detail:

- The symptom could be specified exactly (Table 2) because the affected party reported calls, modes, and sizes, not impressions.
- The server side could be cleared exactly because the wire capture and the call log speak about bytes and flags, not about whether the system "seemed fine."
- The cause could be isolated exactly because result shapes are enumerable: six tools, six objects, one discriminating key.
- The cure could be verified exactly because the test object carries its own check: 9,341 characters and a SHA-256 that either match on the far side of the boundary or do not.
- The record could be closed exactly because the acceptance criterion was fixed in advance and stated in the same units as the failure.

The generalizable standard has four parts, each grounded in a source or in this incident's evidence. **First, treat payload vocabulary as reserved until proven otherwise.** The envelope's words (`content`, and by extension `structuredContent`, `isError`, `_meta`, and generic protocol words such as `type`, `error`, `data`) are not available as payload names; use domain-qualified names (`record`, `file_content`, `text`) instead. This is the documented remedy in the community thread [2], the Google style guide's rule [11], and the .NET guidance [9], converging from three independent directions. **Second, namespace by prefix, because JSON has no other namespacing.** The prefix is the namespace [11]. **Third, test what the model actually receives, per client, not what the server sent.** Practitioner conformance work on MCP makes the same point in the same terms: a compatibility suite needs two records per test, the wire evidence and the model-visible evidence, because a server can satisfy the specification and still fail a particular client [17][18]. This incident's whole method was that discipline applied by hand. **Fourth, prefer loud failure over silent emptiness.** A client that cannot interpret a result should say so in a form the caller can see; a server whose result would be empty should return an explicit error rather than an object a client can mistake for success. The silent `{}` is the worst output a system can produce, because it is indistinguishable from a true empty result and therefore routes around every detector built on error signals.

There is a cost side, and honesty requires stating it. Building to this level of detail is slower than building to the surface: freezes before changes, independent audits before deploys, hash-identified test objects, acceptance criteria fixed in advance, registers that keep superseded diagnoses visible instead of quietly editing them away. The counterfactual cost is visible in the public record of this same defect, where a vendor-scale integration sat broken and officially undetermined for weeks (Section 8.3.5). Detail at the machine layer is not thoroughness for its own sake. It is the difference between a defect that can be closed in a day and one that cannot be closed at all from the outside.

8.3 The AI in the loop

8.3.1 A failure no log could see

In the vocabulary of the dependability literature, the door's behavior throughout was correct service: valid payloads, error flags false, complete logs. The affected client committed what that literature calls a content failure: the service it delivered at its interface, an empty object, deviated from the function it was asked to perform, without any halt or error [13]. The composite failure is best described by the gray failure model from cloud-systems research: a fault whose defining feature is differential observability, in which one entity is negatively affected while the entity responsible for detection perceives the system as healthy [15]. That is a precise description of this incident's shape. The server perceived health. The client experienced absence. No component owned the gap between those two perceptions, so no detector fired, and under a binary healthy-or-failed monitoring model none could have. The defect existed, in full, only in the space between what the wire carried and what one model was shown.

8.3.2 The failure report was filed by the affected party

The defect entered this system's record through Gippy's own audit: an outside AI evaluating its own ability to do document work, reporting the blockage at step 3 of its own workflow, with call-level specificity (which targets, which modes, which sizes, what arrived instead). The investigators' research found no published precedent for this in the literature surveyed: no documented case of an affected AI system autonomously reporting and scoping its own integration failure. The closest public analogue is instructive precisely because it differs: in the OpenAI community thread, the diagnostic probes run inside the affected client (introspecting the result object, finding its key set empty) were executed at human direction by human engineers [2]. Here, the affected party was itself an auditor with a standing assignment to test the system, and it filed the failure as a work product. The practical consequence is measurable in the record: the failure arrived pre-specified, at machine precision, in a form that could be entered directly into a hypothesis table. Anyone who has triaged a defect from a human report of "it doesn't work" will recognize what that is worth, and Section 8.3.5 weighs it against the counterfactual rather than merely celebrating it.

8.3.3 Diagnosis by structural comparison

The diagnosis itself was an elimination argument over enumerable structure (Section 4), executed by an AI operator working from the affected party's specification, the source at a frozen hash, and a wire capture taken through a second AI's connector, which served as the control client. Two features of that process are worth recording. First, the control was real: Muse's connector reads the text block literally, so its successful reads at every size bounded the failure to the affected client's interpretation, the same differential the community testers established with Codex [2]. Second, the process was not immune to the failure mode the literature predicts for AI diagnosis: the investigation's own first hypothesis (size, H1) was wrong, was acted on in the v3 period, and was falsified by the affected party's retests. It survives in this report and in the register because the method requires it to. The measured literature on AI root-cause work supports exactly this posture: AI systems show their value in triage speed, retrieval coverage, and candidate generation, with a human or a test as verifier, as in the large-scale incident studies at Microsoft, where LLM recommendations were evaluated over tens of thousands of production incidents as drafts for on-call engineers [19], and in RCACopilot, whose root-cause category accuracy reached 0.766 micro-F1 while macro-F1 of 0.533 exposed how much weaker rare categories remain [20]. Hypothesis generation is inexpensive and fast; the discipline is in the elimination.

8.3.4 Acceptance by the affected party

The acceptance test closed the loop in the same place it opened: the affected party re-ran the failing operation in its own runtime and reported character counts, slice counts, offset continuity, and hash consistency (Section 7.2). No party to this incident had to take another party's word for the fix. The operator did not declare success from server logs; the vendor did not declare it from a changelog; the affected AI measured it. In a system whose participants are AIs, this is the scalable form of verification: the consumer of a service is the authority on whether the service arrived.

8.3.5 The honest counterfactual: without AI in the loop

How does this class of defect get found and fixed when no AI is a participant? The public record answers with unusual completeness, because the same defect ran in public, at vendor scale, in the same weeks. Human engineers found it: the community thread opens 2026-10-16 with controlled server-side tests (stdio reproduction, a 4,445-character correct payload), gains independent reproductions across Nextcloud and Linear the same day, establishes the Codex-versus-ChatGPT differential by 2026-10-17, produces a first workaround (duplicate the payload as `structuredContent`) by 2026-10-18, and identifies the `content`-key trigger and the rename remedy by 2026-10-22 [2]. That is good engineering, and this report's diagnosis converged on the same finding independently. But the counterfactual has a second half. As of 2026-10-28 the defect was still fully reproducible against the official Linear connector, whose users cannot apply the server-side remedy at all; the vendor's root cause remained, in the thread's word, undetermined; and a support case stood open with no official statement [2]. The human timeline to the workaround was about six days across three vendors and many testers; the timeline to an authoritative explanation is, at last report, unbounded. And the thread's participants could only fix their own servers: the party with the defect, the client vendor, is the one party the process could not reach.

Measured against that baseline, what the AI participants changed in this instance is specific and limited, and it is not magic. **Latency**: report, diagnosis, correction, and acceptance completed inside one day, most of it inside one afternoon, because the reporting party, the operator, and the accepting party were all continuously present in the same record. **The locus of the failure report**: the defect was reported by the system that experienced it, at machine precision, eliminating the reproduction-and-triage phase that consumed the community thread's first days. **The acceptance authority**: closure was granted by the affected party's own measurement rather than by the fixer's assertion. What AI did not change is equally specific. The evidentiary work was the same work: wire capture, source inspection, hypothesis elimination, independent audit, point-of-use acceptance. The wrong first hypothesis still happened. The mechanism is still unconfirmed, exactly as it is for the human engineers, because the client is closed to both. And the published evidence on AI diagnosis argues against any stronger claim: studies of LLM-based program repair find hallucinated, ungrounded patches in a large share of cases, including patches that pass the available tests [21], and benchmark work on agentic root-cause analysis finds low exact-match accuracy with hallucinated data interpretation as a dominant, capability-resistant pitfall (preprint) [22]. The classic automated-repair literature made the underlying point a decade before LLMs: a patch that makes the symptom disappear under a weak test is not a cure, and can be worse than the disease [23]. This incident contains its own small instance: the community's first workaround, duplicating the payload into `structuredContent`, makes the symptom vanish while leaving the client defect untouched and doubling the wire size; the rename, by contrast, is defensible because it removes the indicted property and was verified against the protocol's own terms, which permit text-only results. The difference between those two remedies is not AI versus human. It is evidence versus plausibility, and it is available to any investigator, carbon or silicon, who insists on the wire.

The owner's formulation, which this report adopts as its conclusion on the point, is that the system had to be pushed to this level because the AI story is not incidental to the defect: the failure lived inside an AI client, was reported by an AI auditor, was diagnosed by an AI operator against an AI control, and was accepted by the affected AI's own retest. Without AI participants, the defect would still have been found, eventually, by exactly the kind of public, multi-vendor, multi-week process the record shows. What would not have existed is a closed loop inside one day and one record, with the failure and its cure stated in the same units by the parties who experienced them.

9. Open items and limitations

Stated plainly, in the same labels as the rest of the report:

1. **The client-side mechanism is INFERRED and unconfirmed.** No party outside the client vendor can observe why a top-level `content` key in the payload causes the result to be surfaced as `{}`. The vendor has published no root-cause statement, in this case or in the public thread [2]. If the vendor changes its result handling, the behavior this report describes may change with it, in either direction; the record assembled here (symptom, trigger, cure, acceptance) is what a future investigator would need to re-diagnose from.
2. **Whether other payload keys trigger the same drop is UNKNOWN and untested.** The public record and this incident test `content` only. The broader caution list in Section 8.2 (`structuredContent`, `isError`, `_meta`, `type`, `error`, `data`) is precautionary synthesis from the reserved-name precedents, not documented fact about this client.
3. **Whether the vendor has since changed the behavior is UNKNOWN.** The last public data point in the thread is 2026-10-28 (still reproducible, vendor case open). This system's acceptance on 2026-11-20 proves the rename works against the client as it behaved that day; it does not prove the original key would still fail today, and no claim in this report depends on that.
4. **The acceptance covers one client, one tool, one document.** Generalization to other ChatGPT surfaces (widget-rendered apps, the Responses API) is not claimed; the documented divergence among clients (Section 8.1) is itself the reason not to generalize from a single consumer.
5. **A residual design exposure remains on the server side.** The door still declares no `outputSchema`, and no payload-key convention was codified beyond the warning comment at the change site and the register's standing lesson (payload keys must never reuse the envelope's field names; prefer `record`). Codifying the convention in the door's tool-authoring path, and adding a regression test that runs a payload containing envelope vocabulary through each supported client, are recommended and not yet done.
6. **The fallback correction remains unbuilt, by design** (Section 6.2). If a future client regression defeats the rename, the fallback is the prepared next step, and this report's evidence chain is the basis on which it would ship.

10. Conclusions

1. Defect D-20 was a reserved-name collision at a protocol boundary: the only tool result in the system carrying a top-level `content` key was the only tool whose results an external client silently discarded. The singularity is KNOWN; the client's internal mechanism is INFERRED and remains unconfirmed by its vendor.

2. The correction was one word, `content` to `record`, in one tool, with the rest of the system held provably constant by freeze, independent audit, and hash-guarded deploy. Minimal corrections are not a stylistic preference; they are what make a cure attributable to a cause.
3. The defect was closed by the only test that could close it: the affected party's own successful read, in its own runtime, of a hash-identified document, against an acceptance criterion fixed in advance. Server-side verification was necessary and was not sufficient, which is the end-to-end structure of every defect of this class.
4. The class is established and named in the literature and in protocol practice: namespace collision, reserved-name collision, shadowing. Mature systems reserve vocabulary and rename payloads. The durable rule for builders at any MCP or API boundary is the one this incident cost a day to prove: a name is structure, envelope vocabulary is not available, and silence is the failure mode to design against.
5. AI participants compressed the lifecycle of this defect from the public baseline of weeks to one day, by relocating the failure report to the affected system itself and the acceptance to the affected system's own measurement. They did not relax any evidentiary requirement, and the one they were most tempted to skip, a plausible first hypothesis, was wrong. The level of detail is the story: systems that keep their machine layer exact enough to be checked against their surface are systems whose defects can be found, fixed, and closed by whoever, or whatever, is on duty.

References

[1] Model Context Protocol, Schema Reference, `CallToolResult` definition, specification revision 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18/schema

[2] OpenAI Developer Community, "ChatGPT MCP file reads return empty objects across Gitea and Nextcloud integrations," thread begun 2026-10-16, updates through 2026-10-28 (content-key finding and rename confirmation reported 2026-10-22; vendor case #17440919). https://community.openai.com/t/chatgpt-mcp-file-reads-return-empty-objects-across-gitea-and-nextcloud-integrations/1406893

[3] SEP-1624, "Clarify `structuredContent` vs `content` Usage Guidance," modelcontextprotocol/modelcontextprotocol issue 1624, proposal, 2025-10-07. https://github.com/modelcontextprotocol/modelcontextprotocol/issues/1624

[4] OpenAI, "Building MCP servers for plugins and API integrations" (MCP guide: dual-emit `structuredContent` and JSON in `content`, declare `outputSchema`; the guide's fetch example names the full-text field `text`). https://developers.openai.com/api/docs/mcp

[5] OpenAI Developers, "15 lessons learned building ChatGPT Apps" (channel model: `structuredContent` for widget and model, `content` as optional narration, `_meta` widget-only). https://developers.openai.com/blog/15-lessons-building-chatgpt-apps

[6] openai/openai-apps-sdk-examples, issue 144: `content` field not forwarded to the model when a widget is rendered, 2025-12-09. https://github.com/openai/openai-apps-sdk-examples/issues/144

[7] J. Longshore, "MCP structuredContent: Claude Code Drops Your Text Blocks," DEV, 2026-11-14. https://dev.to/jeremy_longshore/mcp-structuredcontent-claude-code-drops-your-text-blocks-7ja

[8] OpenAI Developer Community, "Responses API drops embedded MCP resources and structuredContent," 2026-11-17. https://community.openai.com/t/responses-api-drops-embedded-mcp-resources-and-structuredcontent/1417340

[9] Microsoft Learn, "Breaking change: System.Text.Json checks for property name conflicts" (.NET 10; `$type`, `$id`, `$ref` reserved; recommendation to avoid or rename conflicting property names). https://learn.microsoft.com/en-us/dotnet/core/compatibility/serialization/10/property-name-validation

[10] GraphQL Specification, September 2025 edition, "Reserved Names" (names beginning `__` reserved to the introspection system). https://spec.graphql.org/September2025/

[11] Google JSON Style Guide, "Reserved Property Names" and "Naming Conflicts" ("There is no concept of JSON namespacing... resolved by choosing a new property name"). https://google.github.io/styleguide/jsoncstyleguide.xml

[12] Protocol Buffers Language Guide (editions): field numbers 19000 to 19999 reserved for the implementation; `reserved` declarations for deleted fields. https://protobuf.dev/programming-guides/editions/

[13] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions on Dependable and Secure Computing 1(1), 2004, pp. 11-33. doi:10.1109/TDSC.2004.2

[14] RFC 8259, "The JavaScript Object Notation (JSON) Data Interchange Format," Section 4 (object names SHOULD be unique; behavior with duplicates unpredictable, in practice silent last-wins). https://www.rfc-editor.org/rfc/rfc8259

[15] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, R. Yao, "Gray Failure: The Achilles' Heel of Cloud-Scale Systems," HotOS '17, 2017. doi:10.1145/3102980.3103005

[16] J. H. Saltzer, D. P. Reed, D. D. Clark, "End-to-End Arguments in System Design," ACM Transactions on Computer Systems 2(4), 1984, pp. 277-288. doi:10.1145/357401.357402

[17] OpZero, "How to Debug MCP Connectors Across ChatGPT, Claude, Grok, and More" (client-compatibility matrix; evidence graded observed, documented, inferred), 2026. Practitioner source. https://www.opzero.sh/blog/mcp-conformance-needs-more-than-the-spec

[18] Agenhood, "MCP Compatibility Test for Claude Code and Codex" (two records per test: wire evidence and model-visible evidence), 2026. Practitioner source. https://agenhood.dev/blog/mcp-compatibility-test

[19] T. Ahmed et al., "Recommending Root Cause and Mitigation Steps for Cloud Incidents using Large Language Models," ICSE 2023; study over more than 40,000 production incidents, Microsoft. https://www.microsoft.com/en-us/research/blog/large-language-models-for-automatic-cloud-incident-management/

[20] Y. Chen et al., "Automatic Root Cause Analysis via Large Language Models for Cloud Incidents" (RCACopilot), EuroSys 2024. doi:10.1145/3627703.3629553 (root-cause accuracy up to 0.766 micro-F1, 0.533 macro-F1)

[21] Y. Cai et al., "Better Understanding, Better Fixes? A Study of Hallucination in LLM-based Automated Program Repair," arXiv preprint, 2026. (Reported: repair hallucinations in 72.7% of sampled cases, including patches passing all available tests.) Preprint, not peer-reviewed.

[22] Kim et al., "Why Do AI Agents Systematically Fail at Cloud Root Cause Analysis?" (OpenRCA benchmark), arXiv preprint, 2026. Preprint, not peer-reviewed.

[23] E. K. Smith, E. T. Barr, C. Le Goues, Y. Brun, "Is the Cure Worse than the Disease? Overfitting in Automated Program Repair," ESEC/FSE 2015.

Appendix A: Change record

Item	Value
Defect	D-20, AtlasClose Chat door, `chat_read_file` return path
Failure reported	2026-11-20, by Gippy's external audit (the main thread, entries 287, 301, 302, 304)
Priority	Moved to head of queue by the owner, 2026-11-20, ahead of the front door build
Door before	v3, sha256 `777141db...`
Door after	v4, sha256 `d9a6500b...`
The change	Payload key `content` renamed to `record` in `chat_read_file` only, whole and sliced results; patched file retained in the operator's workspace
Rollback	Single-file backup on host (hash verified to v3); full freeze below
Freeze	Pre-change freeze on the host, 4,404 files, manifest sha256 `d06a4928...`
Freeze archive	The freeze as a single archive file, 363,458,560 bytes, sha256 `e88dc3dc...`
Audit	Independent instance, PASS, no blockers, before deploy; conditions in Section 6.3
Deploy	Hash-guarded install (aborts unless live file is v3); door service only restarted; board unchanged (sha256 `6e0058ab...`), board HTTP 200 after restart
Document under test	Post 3 article, "The Answer to AI Is More AI, Not Less," thread entries 250 and 260, 9,341 characters, sha256 `286656c7...`
Thread record	Entry 308: technical fix report filed. Entry 310: retest instructions posted. Entry 312: acceptance and header reconciliation entered. Entry 313: the pre-change freeze recorded by the Secretary's office
Operator verification	Whole read under `record`, 9,341 characters, hash match; four slices tile exactly; entry 260 identical; `BAD_RANGE` intact; `tools/list` byte-identical
Acceptance evidence	Affected party's own retest in its ChatGPT runtime: 9,341 of 9,341 characters, 4 of 4 slices, offsets consecutive, final slice complete; observed and photographed by the owner at about 13:12 PT, 2026-11-20
Register state	D-20: FIXED AND ACCEPTED 2026-11-20 (door v4). Standing lesson recorded: a tool result's payload keys must never reuse the envelope's own field names (`content`); prefer `record`
Side finding, reconciled	Entry 250's internal header line reads "NOT posted" (status line as filed, immutable); the article was posted to LinkedIn and X on 2026-11-19 and verified live at both (Section 7.2)

Appendix B: Result shapes, before and after

Envelope (unchanged in both versions; the payload rides as a JSON string in the text block):

```
{ "content": [ { "type": "text", "text": "<payload JSON>" } ], "isError": false }
```

Payload, `chat_read_file`, door v3 (failing):

```
{ "thread_id": "...", "entry": 250,
  "filename": "...",
  "sha256": "286656c7...",
  "ext": "md", "size": 9341,
  "content": "<9,341 characters of document text>" }
```

Payload, `chat_read_file`, door v4 (accepted):

```
{ "thread_id": "...", "entry": 250,
  "filename": "...",
  "sha256": "286656c7...",
  "ext": "md", "size": 9341,
  "record": "<9,341 characters of document text>" }
```

Payload, `chat_read_file` sliced, door v4:

```
{ "thread_id": "...", "entry": 250, "offset": 9000,
  "returned_chars": 341, "total_chars": 9341, "more": false,
  "record": "<final 341 characters>" }
```

Control shape, `chat_read_chat` (unchanged; passed in the affected client throughout):

```
{ "record": "<chat record as text>" }
```

End of report.