From Conversation to Project Core

Generating durable operational memory from captures, decisions, work, and evidence

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

From Conversation to Project Core

Abstract

Project memory in contemporary AI systems is usually implemented as transcript history, retrieved snippets, generated summaries, or opaque vendor memory. These forms preserve language but often fail to preserve operational distinctions: a proposal versus a decision, a worker return versus verification, a completed tool call versus human acceptance, an obsolete specification versus the active one.

Generated Operational Memory is the process of compiling raw conversation, files, tool events, and human decisions into typed durable project state. The target object is a **Project Core**: an event-sourced record of accepted truth, plans, constraints, permissions, artifacts, work orders, returns, evidence, receipts, dependencies, and unresolved questions. Models receive scoped views of the Core and may propose patches; no provider transcript becomes the sole authority.

The architecture preserves immutable capture, explicit promotion of intent, non-destructive accepted and working results, dependency-driven staleness, and reopen-from-state behavior. It treats memory generation as a governed compilation problem rather than a summarization feature.

1. Why summaries are not enough

A project summary can be fluent and still operationally wrong.

It may say:

- “the user wants X” when X was an assistant suggestion;
- “the build is complete” when a worker only returned files;
- “the current design is blue” because an obsolete mockup was retrieved;
- “tests passed” because a report contained that phrase;
- “the project uses model A” after the adapter changed;
- “the user accepted the artifact” when the user merely viewed it.

The failure is caused by missing types and transitions. The system remembers text but not the institutional meaning of the text.

Generated Operational Memory asks the compiler to identify durable objects and relationships rather than produce one narrative.

2. Immutable Capture

Every input first becomes a Capture:

```
capture_id
project_id
source_type
exact_payload_ref
speaker_or_source
timestamp
privacy
ingest_context
```

Capture may be voice, text, image, file, link, email, tool output, webhook, or model return.

The exact input is preserved before interpretation. Corrections create new captures or annotations. This supports later reinterpretation and protects the user's words from being silently replaced by a model's cleaned version.

Capture is not canon. It is evidence that something was provided.

3. Interpretation and intent boundary

A compiler may propose:

- facts;
- preferences;

- goals;
- constraints;
- decisions;
- tasks;
- questions;
- sources;
- permissions;
- contradictions;
- artifact references.

The proposed interpretation is shown against the source. The user or an authorized standing rule may promote selected items into the Project Core.

The critical boundary is:

> Conversation must cross a visible boundary before becoming intent.

This prevents a casual brainstorm, joke, emotional reaction, or assistant suggestion from silently directing consequential work.

4. Project Core entities

A Project Core can include:

- `Project`;
- `ProjectEvent`;
- `Capture`;
- `Interpretation`;
- `Decision`;
- `PlanRevision`;
- `Constraint`;
- `Question`;
- `PermissionGrant`;
- `ProviderBinding`;
- `WorkOrder`;
- `WorkerRun`;
- `Return`;
- `Evidence`;
- `Verification`;
- `Acceptance`;
- `Artifact`;
- `ArtifactVersion`;
- `Dependency`;
- `Receipt`;
- `CostEstimate`;
- `PublicationProposal`.

The list is not valuable because it is long. It is valuable because the objects prevent category collapse.

5. The operational lifecycle

A real project loop is:

```
Capture
→ proposed interpretation
→ source diff
```

```
→ explicit authorization
→ accepted Plan
→ bounded Work Order
→ execution
→ Return
→ Evidence
→ Verification
→ accept / revise / reject
→ Receipt
→ projection update
→ close and reopen
```

Each arrow is an event. Several may occur automatically under standing policy, but the policy and scope remain visible.

The distinctions are strict:

- completed is not accepted;
- Return is not verification;
- verification is not acceptance;
- deployment is not production seal;
- publication requires a separate proposal and authority.

6. Accepted versus working results

A project maintains separate references:

```
accepted_result_ref
working_result_ref
candidate_refs[]
```

The worker may continuously improve the working result. It cannot erase the accepted result. A new candidate can fail, regress, or be abandoned without destroying the last accepted state.

Promotion is a governed event with evidence and human acceptance where required.

This is the Project Core equivalent of version control safety without forcing the user to understand Git.

7. Work Orders and Returns

A Work Order is a bounded request:

```
work_order_id
objective
inputs
constraints
allowed_tools
authority
budget
deadline
expected_outputs
evidence_contract
failure_behavior
return_schema
```

A Return contains:

```
output_refs
changed_objects
logs
cost
limitations
failures
evidence_refs
worker_claimed_status
```

The runtime verifies the Return independently. A model or worker cannot self-certify acceptance.

8. Dependencies and staleness

Artifacts and plans declare dependencies.

When a canonical input changes, the system can identify:

- unaffected objects;
- stale objects;
- objects requiring deterministic refresh;
- objects requiring model regeneration;
- objects requiring human review.

This is generated operational memory because it records not only what exists but what the current change means for the work graph.

9. Provider independence

ChatGPT, Claude, Codex, local models, GitHub, and specialist tools are bindings to one Project Core.

A provider card is a binding, not a copy. The provider receives:

- selected project;
- scoped objective;
- exact evidence;
- authority;
- output contract;
- relevant accepted state.

The provider returns a proposal or artifact. Durable state remains outside the provider transcript.

Changing providers does not discard the project. Model-specific caches can be invalidated while project truth remains.

10. Generated memory planes

The system should distinguish:

- **personal:** user-owned preferences and standing instructions;
- **project:** accepted state for one project;
- **working:** temporary active context, candidates, and scratch;
- **procedural:** reusable methods, skills, and runbooks;
- **public:** explicitly promoted proof and publications.

Movement between planes requires a proposal and authority. Private conversation does not become public training data or registry state by default.

11. Reopen as a proof

The decisive operational test is not whether the app can display a recent transcript. It is whether a project can be closed, the process terminated, and the exact work state reconstructed.

On reopen, the system should recover:

- accepted objective;
- active Plan;
- constraints;
- permissions;
- accepted and working artifacts;
- pending Work Orders;
- unresolved questions;
- Returns awaiting verification;
- stale dependencies;
- latest receipts;
- next valid action.

If the project requires rereading the old conversation to know what is happening, the Project Core is incomplete.

12. Zeke's always-working loop

The orchestration layer can continually ask:

> What is the smallest authorized action that would reduce the gap between the active Plan and the current working result?

Safe background work may include:

- indexing attachments;
- detecting stale dependencies;
- drafting Work Orders;
- estimating cost;
- deterministic validation;
- creating proxies;
- organizing evidence;
- surfacing contradictions;
- compiling context.

Without authority it may not spend, publish, send messages, change permissions, merge protected work, expose private data, or treat worker completion as acceptance.

13. Evidence state

The Project Core doctrine is specified across Glyph Desk, Glyphd V1, Continuity Core, LCSM, and accepted-act build directives. Existing related systems provide partial implementation evidence: local process directors, project stores, receipt schemas, deterministic build engines, and registry proof records.

No single production repository is represented here as a fully integrated universal Project Core. The paper therefore distinguishes a well-developed architecture from a completed

platform implementation.

14. Evaluation

Continuity

- reopen accuracy;
- accepted constraint retention;
- obsolete-state exclusion;
- source/proposal distinction;
- provider handoff accuracy;
- time to resume.

Governance

- unauthorized action count;
- stale approval rejection;
- cross-project isolation;
- acceptance separation;
- permission revocation;
- publication boundary.

Operations

- Work Order completeness;
- Return normalization;
- evidence coverage;
- deterministic verification;
- dependency refresh;
- rollback;
- idempotency.

Human workload

- number of decisions re-explained;
- time spent finding latest state;
- branch confusion;
- trust in worker status;
- comprehension of what remains.

15. Security and privacy

Threats include:

- prompt injection promoting source text into authority;
- assistant proposal recalled as user decision;
- cross-project context leakage;
- stale permission reuse;
- private capture exported to cloud;
- worker Return promoted to fact;
- approval replay;
- hidden substitution;
- malicious connector.

Controls include immutable Capture, typed authority, scoped bindings, exact source refs, revocable permissions, append-only events, validator gates, redaction, local-first paths, and receipts.

16. Limitations

Operational memory can become burdensome if every thought must be promoted through ceremony. The interface needs proportional structure and good defaults.

The compiler may misclassify intent. A typed graph can still encode the wrong interpretation. Human correction and source diff remain essential.

Event-sourced state can be complex to migrate and query. Projections, indexes, and compaction require careful engineering without destroying history.

17. Conclusion

Generated Operational Memory is not a better summary.

It is the conversion of conversation and work into durable project objects with identity, authority, evidence, dependencies, and state transitions.

The durable object is the Project Core. Models rent scoped context. The user owns the project.

Architecture illustration briefs

- **P08-F01 — From Capture to Project Core**: Raw captures enter interpretation, source diff, authorization, then typed project objects; transcript remains archive.
- **P08-F02 — Project operational lifecycle**: Capture → Plan → Work Order → Return → Evidence → Verification → Acceptance → Receipt, with divergence states.
- **P08-F03 — Accepted and working references**: Immutable accepted artifact beside evolving working branch and candidates; promotion is a separate event.
- **P08-F04 — Provider bindings**: Multiple provider surfaces rent scoped views of one Project Core and return bounded outputs.

Source register

- **P08-S01** — Glyph Desk Master Specification v1 (2026-07-12). `GLYPH-DESK-MASTER-SPEC.v1.md`
- **P08-S02** — Glyphd V1 Canon Recovery and Gap Audit (2026-07-18). `ZEKE-V1-CANON-RECOVERY-AND-GAP-AUDIT-v0.1.md`
- **P08-S03** — Continuity Core Design Review 2 (2026-07-21). `CONTINUITY-CORE-DESIGN-REVIEW2.md`
- **P08-S04** — Native Accepted Act build directive (2026-07-20). `Pasted text.txt`
- **P08-S05** — LCSM Workbench E2E Specification (2026-07-04). `LCSM-WORKBENCH-E2E-SPEC.v0.md`

Public-disclosure and IP boundary

This edition publishes the public-safe Project Core model and operational distinctions. It excludes private user captures, unpublished internal compiler prompts, production credentials, customer schemas, and confidential patent drafting. It does not assert that every Glyphd product already shares one integrated Project Core.