Continuity Core

Concept identity, automatic checkpoints, project resurrection, and source-authority separation

Hayden Lindley · Glyphd Labs

2026-07-27

Contents

Continuity Core **1**
Abstract . . . 1
1. The continuity failure . . . 2
2. Stable concept identity . . . 2
3. Canon is event sourced . . . 3
4. Source and authority are separate . . . 3
5. Automatic checkpoints . . . 4
6. Project resurrection . . . 4
7. Bindings, not ownership theft . . . 4
8. Vocabulary migration . . . 5
9. Checkpoint service and Return inbox . . . 5
10. Continuity across providers . . . 5
11. Relationship to LCSM, GlyphCAS, and Project Core . . . 6
12. Security and privacy . . . 6
13. Evaluation . . . 6
14. Current evidence state . . . 7
15. Limitations . . . 7
16. Conclusion . . . 7
Source register . . . 7
Revision history . . . 7

Continuity Core

Abstract

Serious human–AI work rarely remains inside one application. It moves among conversations, repositories, documents, design tools, model providers, local machines, deployments, and human memory. The resulting failure is not merely “the AI forgot.” The project itself lacks a durable identity and a reconstructable account of what became true.

Continuity Core is an account-level substrate for stable concept identity, event-sourced canon, aliases, supersession, automatic checkpoints, provenance, and project resurrection. It does not replace product-local state. It provides the continuity layer through which products can know that two names refer to one evolving concept, that a proposal did not become a founder decision, that a build superseded an earlier plan, and that a dormant project can be reopened without re-deriving months of work.

Its central rule is:

> **A provider conversation is a source. It is not the project.**

1. The continuity failure

The same concept may appear as:

- a chat thread;
- a markdown specification;
- a renamed repository;
- a design mockup;
- a prototype;
- a deployment;
- an agent Return;
- a founder correction;
- a status entry;
- an archived predecessor.

Without stable identity, every rename fragments lineage. Without source labels, model suggestions are later recalled as user decisions. Without checkpoints, topic switches drop state. Without a resurrection protocol, a dormant project wakes as a search problem.

These are account-level failures. Solving them independently in every app creates competing memory systems.

2. Stable concept identity

Continuity Core assigns a stable `concept_id` independent of product name, repository path, or display label.

```
Concept {
  concept_id
  canonical_name
  aliases[]
  status
  parent_or_family
  source_records[]
  accepted_claims[]
  open_conflicts[]
  implementation_bindings[]
  lineage_edges[]
}
```

A rename appends an alias and, where appropriate, a dated naming decision. It does not create a new concept by default.

A split, fork, merge, or absorption uses explicit lineage edges:

```
RENAMED_FROM
SUPERSEDES
FORKED_FROM
MERGED_INTO
ABSORBED_BY
DERIVED_FROM
IMPLEMENTED_BY
PROPOSED_AS
REJECTED_IN_FAVOR_OF
```

The graph preserves history without forcing old public names to remain active.

3. Canon is event sourced

Continuity Core does not overwrite project history. Material changes are events:

```
capture.recorded
proposal.created
decision.accepted
decision.corrected
concept.renamed
concept.forked
repo.bound
artifact.created
artifact.accepted
implementation.verified
claim.downgraded
project.checkpointed
project.dormant
project.resurrected
```

Current state is a projection over these events. Corrections supersede prior state without pretending the prior event never happened.

4. Source and authority are separate

Every statement records what kind of thing it is.

```
statement_text
source_ref
speaker_or_agent
authority_class
status
scope
evidence_refs
recorded_at
supersedes?
```

Authority classes distinguish:

- raw user capture;
- assistant proposal;
- founder decision;
- repository implementation truth;
- test result;
- external documentation;
- inferred relationship;
- disputed or stale statement.

A model proposal may be useful and memorable. It does not become founder intent until an acceptance event exists.

This is one of the most important protections in an AI-native project system because conversational fluency otherwise erases provenance.

5. Automatic checkpoints

A checkpoint is a compact, machine-readable state bundle produced at meaningful boundaries:

- session end;
- topic switch;
- model/provider handoff;
- branch merge;
- deployment;
- acceptance;
- prolonged inactivity;
- manual “save the state” action.

A checkpoint contains:

```
project_id
head_event
accepted decisions
active plan
working result
accepted result
open work orders
blocked items
known conflicts
source and repo bindings
latest receipts
next useful actions
checkpoint_digest
```

The checkpoint is derived from canonical events. It is not a new competing truth store.

6. Project resurrection

A resurrection protocol answers:

1. What is this project?
2. What is its stable identity and naming history?
3. What has been accepted?
4. What is merely proposed or speculative?
5. What implementation exists, and where?
6. Which evidence is current?
7. What is blocked?
8. What was the last known-good state?
9. What is the smallest useful next action?

The system gathers the latest accepted checkpoint, verifies bindings, detects stale dependencies, and renders a Resurrection Surface. Unknowns remain explicit.

A project should not need a human to reread fifty chats merely to restore basic orientation.

7. Bindings, not ownership theft

A repository, Vercel deployment, Google Drive file, or provider chat is a binding to the concept. None automatically owns the Project Core.

```
Binding {
  binding_id
```

```
  concept_id
  kind
  external_identity
  role
  authority
  freshness
  last_verified
  evidence_ref
}
```

Bindings can become stale or be superseded. The underlying concept identity remains.

8. Vocabulary migration

Naming changes are ordinary events. A vocabulary migration can:

- add new canonical label;
- preserve historical aliases;
- mark prohibited or retired public names;
- rewrite active projections;
- retain source wording in archived material;
- validate that references resolve.

This turns repeated manual cleanup into system behavior.

9. Checkpoint service and Return inbox

A Universal Return Inbox receives email, webhooks, Git events, workers, files, and provider outputs. Intake preserves exact provenance and does not promote incoming material directly into canon.

The Checkpoint Service evaluates material events and refreshes project projections. A Return may update working state, add evidence, or propose a decision. Human acceptance remains a separate transition.

10. Continuity across providers

Provider connectors receive a scoped snapshot or PromptCapsule. They do not receive ownership of the project and do not need the entire historical transcript.

A handoff records:

```
provider
project_id
scope
source refs
accepted decisions
active objective
constraints
privacy class
response or Return
```

A later model can continue from the same Project Core without simulating the prior provider's conversation.

11. Relationship to LCSM, GlyphCAS, and Project Core

Continuity Core is an account- and portfolio-level application of the same doctrine:

- **LCSM** supplies append-preserving accepted state and provenance.
- **Project Core** supplies the durable object for one project.
- **GlyphCAS** supplies exact identity, semantic aliases, dependencies, and reuse.
- **Continuity Core** resolves cross-project concept identity, checkpoints, bindings, resurrection, and vocabulary lineage.

It should not duplicate application databases or turn every artifact into a public registry entry.

12. Security and privacy

Continuity increases the blast radius of mistakes unless scopes are strict. Threats include cross-project leakage, authority confusion, stale source promotion, malicious connector returns, alias hijacking, and accidental public projection.

Required controls:

- project-scoped encryption and access;
- explicit cross-project grants;
- source/authority separation;
- immutable event records;
- connector scopes;
- private-by-default checkpoints;
- publication proposals distinct from acceptance;
- revocation and export;
- no silent whole-history transfer.

13. Evaluation

Benchmark project recovery under controlled fragmentation:

- rename and alias chains;
- conflicting model proposals;
- accepted and working result divergence;
- stale repositories;
- missing connector;
- dormant periods;
- provider switches;
- partial checkpoints;
- corrupted cache with intact event log.

Measure:

- time to accurate orientation;
- accepted-decision precision and recall;
- proposal-as-decision errors;
- stale-binding detection;
- restart/reopen accuracy;
- cross-project leakage;
- required human reconstruction effort;
- replay agreement.

14. Current evidence state

The corpus includes a detailed Continuity Core design review, Project Core contracts, canon recovery audits, repository bindings, receipt doctrine, and repeated firsthand incidents motivating each gap. The integrated account-level service remains `SPECIFIED`; partial mechanics exist across Command Center, Glopper, registry, project specs, and build systems.

15. Limitations

Identity resolution can make false merges. Excessive checkpointing creates noise. A global service can become overcentralized. Historical sources may conflict. Automated summaries can omit nuance. Therefore, exact sources remain available, merge operations are reviewable, and unresolved identity remains unresolved rather than forced.

16. Conclusion

Continuity Core turns memory from “whatever a model recalls” into an inspectable relationship among stable concepts, exact sources, authority, events, checkpoints, and implementations. Its value is not infinite memory. Its value is reliable return: the ability to stop, switch tools, rename, revisit, and continue without losing what the work became.

Source register

- `CONTINUITY-CORE-DESIGN-REVIEW2.md`
- `ZEKE-V1-CANON-RECOVERY-AND-GAP-AUDIT-v0.1.md`
- `GLYPH-DESK-MASTER-SPEC.v1.md`
- Current project-control and receipt doctrine cited by those sources

Revision history

- **0.1.0 — 2026-07-27:** Initial public-safe institute edition.