A Roadmap for Implementing Utilities in Spherepop Calculus

Flyxion

December 13, 2025

1 Purpose

This document presents a conservative, implementation-oriented roadmap for building user-facing utilities atop the Spherepop OS kernel. The focus is not on feature completeness but on preserving the core invariants: deterministic replay, total causal order, ABI stability, and strict separation between authoritative semantics and derived views.

Utilities are treated as structured consumers and producers of events and views, not as stateful programs with implicit side effects.

2 Design Constraints

All utilities must satisfy the following constraints.

2.1 Authoritative Discipline

Utilities may not mutate kernel state directly. All authoritative changes must be expressed as proposed events submitted to the arbiter. Utilities may generate speculative overlays, but these overlays must remain explicitly non-authoritative.

2.2 Replay Compatibility

Utilities must not rely on hidden state, ambient context, or execution timing. Given the same event log prefix and the same utility invocation, the resulting proposals or views must be identical.

2.3 ABI Respect

Utilities interact with the kernel through stable ABI-defined event layouts. No utility may assume undocumented padding, ordering, or reinterpretation of event structures.

2.4 View Non-Interference

Utilities that produce views (diffs, JSON renderings, summaries) must not feed information back into kernel decisions. Views may be dropped or recomputed without semantic consequence.

3 Spherepop Calculus as the Utility Substrate

Spherepop calculus is not a scripting language layered on the OS; it is the algebra of authoritative operations supported by the kernel.

The primitive calculus operations correspond directly to event types:

- `POP`: introduce a semantic object handle
- `MERGE`: induce equivalence between objects
- `LINK`: create a typed relation
- `UNLINK`: remove a typed relation
- `COLLAPSE`: perform bulk equivalence
- `SETMETA`: attach non-semantic metadata

Utilities operate by composing these primitives, never by bypassing them.

4 Utility Taxonomy

Utilities fall into three broad classes.

4.1 Proposal Generators

Proposal generators produce candidate event sequences but do not commit them. Examples include batch object creation, canonicalization tools, and semantic refactoring utilities.

Such tools terminate by emitting a proposal stream, not by mutating state.

4.2 View Generators

View generators consume replayed state and emit derived representations: JSON graphs, diffs, textual summaries, or tabular listings. They are observational only.

4.3 Overlay Managers

Overlay managers manipulate speculative branches: creating overlays, rebasing them, or discarding them. They operate entirely outside the authoritative log.

5 Phase I: Minimal Utility Set

The first phase focuses on utilities that exercise the kernel without expanding it.

5.1 Connection and Replay

- `sp`: connect to an arbiter or log source
- `sp-replay`: replay a log prefix into a local kernel instance

These tools establish the basic contract between utilities and replay.

5.2 Object and Relation Tools

- `sppop`: batch generation of `POP` events
- `splink`: creation of `LINK` events
- `spunlink`: removal of relations

These utilities must demonstrate representative normalization and replay determinism.

5.3 Diff and Snapshot Inspection

- `spdiff`: inspect derived diffs per event
- `spsnap`: serialize full replayed state

These tools validate viewâĂŞcause separation.

6 Phase II: Canonicalization and Refactoring

This phase introduces utilities that perform structured semantic rewrites.

6.1 Merge and Collapse

- `spmerge`: propose `MERGE` events
- `spcollapse`: perform region-level equivalence

These tools must preserve confluence and replay equivalence.

6.2 Preview-Commit Workflow

Utilities should support a preview mode:

1. generate a speculative overlay
2. replay and inspect the result
3. optionally submit the overlay as a proposal

No utility may auto-commit by default.

7 Phase III: Semantic Query and Analysis

7.1 Query as View

- `spgrep`: query relations and metadata
- `sppath`: explore relation paths

Queries must be expressed as pure functions over replayed state.

7.2 Summarization

Summaries and compressions are admissible only as derived views. Any lossiness must be explicit and reversible via replay.

8 Phase IV: Composition and Pipelines

Utilities should be composable via standard streams. However, pipelines must preserve the distinction between:

- authoritative proposals
- non-authoritative views

For example, a pipeline may transform a view into a proposal, but the transformation must be explicit and inspectable.

9 Non-Goals

The following are explicitly out of scope:

- implicit inference of semantics
- automatic conflict resolution
- hidden mutable caches
- time-dependent behavior

Such features are incompatible with deterministic replay.

10 Relationship to GNU-Style Tooling

Spherepop utilities resemble GNU tools in composability and narrow scope, but differ fundamentally in that they operate over semantic time rather than mutable files.

The closest analogy is not `sed` or `awk`, but a controlled sequence of rewrite proposals applied to a canonical log.

11 Conclusion

The utility ecosystem of Spherepop OS should grow slowly and conservatively. Each tool must justify its existence by preserving the kernel invariants and by reducing, rather than increasing, semantic ambiguity.

The success of Spherepop utilities is measured not by convenience, but by the degree to which they make time, causality, and meaning explicit.