
Epiphany

OPERATION CATALOG

A companion to the Core Specification

Version 0.1.0 — Phase 2 (Ko representative primitives)

Normative for the operation kinds it defines

Contents

1	About This Companion	2
1.1	Relationship to the Core Specification	2
1.2	Conformance Profiles	3
2	The Catalog Framework	4
2.1	Value-Typed Payloads	4
2.2	Per-Primitive Schema Template	4
2.3	Reduction-Discipline Coverage	5
3	K0 — Representative Primitives	6
3.1	InsertEvent	6
3.2	DeleteEvent	7
3.3	RespellPitch	7
3.4	CreateCrossCutting	8
3.5	ChangeRegionTimeModel	8
3.6	SetUserSystemBreak	9
3.7	DeclareTransaction	9
3.8	ResolveConflict (meta-operation)	10
3.9	UndoTransaction (meta-operation)	10
4	v0 → v1 Payload Migration	11
5	K1 — Framework Slots (Phase 3)	12

About This Companion

The *Operation Catalog* is a companion to the Epiphany Core Specification. It fulfils the open question the core specification raises in its *Operation Catalog Conformance* section (Chapter 6, SEMANTIC OPERATIONS AND CONCURRENT REDUCTION, the `sec:semops:catalog` open question), which states that the catalog “is normative once published; this specification is non-final until the catalog is delivered.” This v0.1 release delivers the catalog *framework* and the **Ko representative primitive set** — the operation kinds the Phase 2 visible slice and binary format actually exercise. The remaining Ko and the full 60–80-primitive catalog are drafted as framework slots (Chapter 5) and completed in Phase 3.

1.1 Relationship to the Core Specification

This companion does not restate the operation framework; it *references* it. The framework — operation identity and stamps, the hybrid-logical-clock monotonicity rule, the dotted-version-vector causal context, the order-independent operation-slot model and equivocation, the canonical reduction order, the four-phase lifecycle, conflict records and the conflict registry, re-anchoring, transactions, forward undo, and the LWW discipline — is the core specification’s Chapter 6. This companion defines, for each operation kind, the *payload schema* and how that kind *instantiates* the framework’s reduction, conflict, undo, and re-anchoring rules.

It also consumes, rather than re-deriving, the **ratified byte-convention baseline** (core specification Chapter 8, BINARY FORMAT COMPANION, requirement `req:format:codec-conventions`, and the CANONICAL BYTE-LAYOUT REFERENCE appendix): little-endian integers, a single discriminant byte per tagged union, `u32` length prefixes on every variable-width leaf, and raw UTF-8 free text. Operation-payload encodings are expressed in terms of that baseline; the literal wire layout of each payload is the Binary Format companion’s (Agent J’s) to pin, in coordination with this catalog.

RATIONALE

The catalog is versioned *independently* of the core specification (independent semver). Operation kinds are added over time; the catalog should evolve — adding primitives, refining conflict cases — without forcing a core specification revision. The core specification changes only when the *framework* changes.

1.2 Conformance Profiles

A **Phase-2 profile** implementation **MUST** implement every primitive in Chapter 3 with the schema, reduction rule, conflict cases, undo semantics, and re-anchoring behaviour defined there. The K1 primitives of Chapter 5 are *unavailable* under the Phase-2 profile: an implementation **MUST** reject (not silently ignore) an operation whose kind is a K1 primitive it does not implement.

The Catalog Framework

2.1 Value-Typed Payloads

Every operation payload in this catalog is **value-typed**: it carries the real graph values its effect introduces, not identifiers that point at values in some ambient graph. An `InsertEvent` carries the whole `Event`; a `RespellPitch` carries the whole `PitchSpelling`; a `CreateCrossCutting` carries the whole tie, slur, beam, or spanner.

RATIONALE

The vo prototype carried *identifier-only projections* — an `InsertEvent` held an `EventId` plus reduction-relevant scalars; a `RespellPitch` held a content-hash *fingerprint* of the new spelling. That was sufficient to make an envelope hashable and to drive reduction, but it is **not durable**: an operation replayed in a fresh context — a backup restore, a cross-tool round-trip — needs the full value, which an identifier-only payload cannot supply. Value-typed payloads are what make an Epiphany document portable.

REQUIREMENT

A value-typed payload field **MUST** be encoded by emitting the field's value under the core specification's canonical value encoding (`req:format:codec-conventions`), framed by a `u32` little-endian length prefix. Decoding **MUST** be the exact inverse and **MUST** reject trailing bytes within the framed region. The encoding introduces no byte layout beyond the ratified baseline: a value's bytes here are byte-for-byte the bytes the whole document codec emits for that value.

2.2 Per-Primitive Schema Template

Each catalog primitive is specified under a fixed six-part template. A new primitive is a schema-fill against this template, not a fresh design:

Payload schema The value-typed fields the operation carries, with their core-specification types.

Canonical encoding The field order and framing, consuming Requirement 2.1. (The literal byte layout is the Binary Format companion's; this catalog fixes the *field set and order*.)

Reduction rule How the operation mutates canonical state when it is reached in canonical reduction order — the preconditions it checks, the objects it mints or tombstones, and the bookkeeping it records.

Conflict cases The conflict records the operation can produce, by `ConflictKind`, and which participant materialises.

Undo semantics The compensating effect of undoing a transaction that contains the operation, under each `UndoPolicy` (`StrictInverse` / `BestEffort` / `Cascade`).

Re-anchoring The behaviour when an object the operation references is tombstoned before or concurrently with it.

2.3 Reduction-Discipline Coverage

The Ko representative set is chosen so that, between them, the primitives exercise *every* reduction discipline the framework defines: position-keyed insert with system-voice promotion; delete-wins with tombstones, tuple compensation, and cross-cutting re-anchoring; field-overwrite with last-writer-wins and structural-field-collision conflicts; set-union creation; structural time-model migration; an LWW advisory; atomic transactions with descriptor precedence; and the two meta-operations (conflict resolution and forward undo). A primitive added later that reuses one of these disciplines inherits its reduction, conflict, undo, and re-anchoring treatment.

Ko — Representative Primitives

This chapter is normative under the Phase-2 profile. Each primitive’s reduction, conflict, undo, and re-anchoring behaviour is the behaviour the core specification’s Chapter 6 defines for its discipline; the description here states how the primitive instantiates it. The reference implementation is `epiphany-ops` (the `payload`, `reduce`, and `migrate` modules).

3.1 InsertEvent

Payload schema. `InsertEventOp { staff_instance: StaffInstanceId, event: Event }`. The voice, region-local position, duration, and pitch identities are read from the `Event` value; the `staff_instance` is retained alongside it so the system-voice promotion derivation is total without a containment walk.

Canonical encoding. `staff_instance`, then the length-framed canonical bytes of `event`.

Reduction rule. A position-keyed insert. Preconditions: the event’s duration is positive; the event id is neither live nor tombstoned; in graph-aware reduction the target voice exists in a metric region and the event’s pitch ids are fresh. The event and its pitches are minted live; the voice is created on first use. Concurrent inserts whose half-open duration intervals overlap in the same voice are resolved by an order-independent promotion pre-pass: the lower-`OperationId` insert is retained in the original voice, and each overlapping loser is promoted to the deterministic system voice `derive_promoted_voice_id(staff_instance, voice, winner, loser)` and tagged `VoicePromoted`.

Conflict cases. None at reduction time; promotion is a deterministic repair, not a conflict.

Undo semantics. Undoing the enclosing transaction tombstones the minted event, its pitches, and any promoted voice. `StrictInverse` conflicts if any was already tombstoned or modified; `BestEffort` tombstones the survivors; `Cascade` is `StrictInverse` over the same minted set (dependent-closure undo is a Phase-3 refinement).

Re-anchoring. Not applicable (the operation mints, it does not reference a pre-existing object that could be tombstoned).

3.2 DeleteEvent

Payload schema. `DeleteEventOp { event: EventId, tuple_compensation: TupleCompensation }`, where `TupleCompensation` is `NotInTuple`, `ReplaceWithRest { rest: Rest }` (value-typed replacement `rest`), `RewriteTuples { tuples }`, or `CascadeDeleteTuples { tuples }`.

Canonical encoding. `event`, then the tuple-compensation discriminant and its payload (a length-framed `Rest` value for `ReplaceWithRest`).

Reduction rule. Delete-wins: the event and its contained pitches are tombstoned (retaining their identifiers). Tuple compensation, when present, adds the replacement `rest` live or tombstones the cascaded tuple group. Concurrent deletes of the same event are idempotent.

Conflict cases. None for the delete itself. Graph-aware reduction refuses an ill-formed tuple compensation as a precondition failure (no-op).

Undo semantics. An insert-shaped compensation re-introduces the tombstoned content; for the prototype’s minted-object model this is the inverse of the tombstone set, with the policy treatment described under `InsertEvent`.

Re-anchoring. Tombstoning the event runs the framework’s re-anchoring rule table over every cross-cutting structure that referenced it: a tie cascade-deletes; a comment or analytical annotation orphans (user content is never silently deleted); a beam truncates while ≥ 2 members survive and otherwise cascade-deletes; a slur or spanner re-anchors to the nearest surviving endpoint while ≥ 1 survives and otherwise cascade-deletes.

3.3 RespellPitch

Payload schema. `RespellPitchOp { pitch: PitchId, spelling: PitchSpelling }` — the full spelling value (`v1`), not a fingerprint.

Canonical encoding. `pitch`, then the length-framed canonical bytes of `spelling`.

Reduction rule. A last-writer-wins field overwrite, keyed by `pitch`. Precondition: the pitch is live. The resolved spelling is the one carried by the operation latest in canonical order. Two respellings of one pitch that are causally ordered overwrite intentionally; two *concurrent* respellings with *equal* spelling reduce idempotently.

Conflict cases. Two concurrent respellings of one pitch with *differing* spellings produce a `StructuralFieldCollision` conflict recording the winner (later in canonical order), the loser, and the field `spelling`. The winner materialises and carries the `Conflicted` effect tag.

Undo semantics. Undo restores the pre-operation spelling (or removes the spelling if the operation introduced the first one), under the active policy.

Re-anchoring. If the target pitch is tombstoned, the respelling is a no-op (`TargetTombstoned`).

OPEN QUESTION

P12-K1. A `vo RespellPitch` carried only a content-hash *fingerprint* of the spelling. The fingerprint cannot be inverted to a `PitchSpelling` without a side table, so the `vo→v1` migration (Chapter 4) recovers the spelling from the score graph context — an explicit per-pitch spelling attachment whose canonical bytes hash to the fingerprint — and, when the context lacks it, declares the envelope unmigratable (the bundle opens read-only). This is the one representative payload that is not self-contained under migration; the disposition (whether a richer `vo` corpus, or a documented read-only fallback, is the long-term answer) is a Pass-12 question.

3.4 CreateCrossCutting

Payload schema. `CreateCrossCuttingOp { structure: CrossCuttingValue }`, where `CrossCuttingValue` is the typed value of a `Tie`, `Slur`, `Beam`, or `Spanner`. Its identity and referenced endpoints are read from the value.

Canonical encoding. A discriminant for the structure kind, then the length-framed canonical bytes of the structure value.

Reduction rule. Set-union creation: the structure is minted live if its id is not already live and every referenced endpoint is live; a second create of a live id is idempotent. Graph-aware reduction materialises the structure with its full fields.

Conflict cases. None (all-or-nothing creation; union is deterministic).

Undo semantics. Undo tombstones the minted structure, under the active policy.

Re-anchoring. The structure participates in the re-anchoring rule table when one of its endpoints is later tombstoned (see `DeleteEvent`).

Migration coverage. The `vo→v1` migration (Chapter 4) reconstructs the event-anchored `Tie`, `Slur`, and `Beam` from the `vo` reference (id plus event endpoints). A `Spanner` is anchored by `TimeAnchors` rather than a fixed pair of event endpoints, so its full value is not reconstructable from the `vo` event-reference projection; a `Spanner`-create is therefore reported unmigratable (read-only) under M1, alongside the respell case of P12-K1. A faithful spanner migration joins when the projection carries the anchors — M2.

3.5 ChangeRegionTimeModel

Payload schema. `ChangeRegionTimeModelOp { region: RegionId, new_time_model: RegionTimeModel, declared_incompatible: Vec<EventId>, remapping: PositionRemapping }` — the full target model value (`v1`).

Canonical encoding. `region`, the length-framed `new_time_model` value, the canonically-ordered `declared_incompatible` set, then `remapping`.

Reduction rule. Structural migration. The region adopts the target time model. Graph-aware reduction derives coordinate-kind incompatibilities from the region's events (and from the remapping coverage) and refuses a migration that would violate the coordinate discipline.

Conflict cases. Concurrent same-region migrations produce a `StructuralFieldCollision` on the field `time_model`; a migration with incompatible events produces a `TimeModelMigrationFailure` naming the region and the incompatible events. A causally-later migration is re-evaluated against the first migration’s graph rather than conflicting.

Undo semantics. Undo restores the region’s prior time model under the active policy.

Re-anchoring. Not applicable.

OPEN QUESTION

P11-C6. The rich migration payload — a coordinate converter rather than a `declared_incompatible` list plus a `PositionRemapping` — remains the catalog’s to design when graph-aware migration is the only reduction path.

3.6 SetUserSystemBreak

Payload schema. `SetUserSystemBreakOp { region: RegionId, anchor: TimeAnchor, present: bool }` — the full anchor value (`v1`).

Canonical encoding. `region`, the length-framed anchor value, then the boolean.

Reduction rule. A last-writer-wins advisory. The break preference is recorded for the region keyed by the anchor’s *resolved musical position*; graph-aware reduction adds or removes the anchor from the region’s user system-break list.

Conflict cases. None (LWW advisory).

Undo semantics. Undo restores the prior advisory value for the `(region, resolved-position)` key.

Re-anchoring. Not applicable in the prototype (the advisory is keyed by resolved position; a tombstoned anchor target degrades to the region origin).

3.7 DeclareTransaction

Payload schema. `TransactionDescriptor { id: TransactionId, label: String, category: Option<TransactionCategory> }`. Value-complete in `vo` and `unchanged`.

Reduction rule. Records the descriptor. Member primitives reference the transaction id and **MUST** causally depend on the descriptor; the members reduce atomically (all-or-nothing) in canonical order.

Conflict cases. A missing descriptor or a member that does not causally follow it produces a `TransactionConflict`; any member failure rolls back the whole transaction and all members read `NoOp{TransactionConflict}`.

Undo / re-anchoring. Transactions are the unit of undo (Section 3.9); re-anchoring is per member.

3.8 ResolveConflict (meta-operation)

Payload schema. `ResolveConflictPayload { target: ConflictId, action: ResolutionAction }`. Value-complete.

Reduction rule. Transitions the target conflict’s resolution state. An action of `Dismiss` reaches the `Dismissed` state; any other action reaches `Resolved`. Re-resolving with the same action is idempotent; two concurrent resolves with differing actions produce a meta-conflict.

RATIONALE

Pass 11 added `ResolutionAction::Dismiss` (item 2.5) precisely so the `Dismissed` state is reachable by an authored operation rather than merely representable. The catalog records that `Dismiss` is the action that selects it (resolving the vo ambiguity P11-C10).

3.9 UndoTransaction (meta-operation)

Payload schema. `UndoTransactionPayload { target: TransactionId, policy: UndoPolicy }`, with `UndoPolicy` one of `StrictInverse`, `BestEffort`, `Cascade`. Value-complete.

Reduction rule. A forward compensating edit computed against the materialised state at the undo’s canonical position (never literal time travel). The prototype models the compensation as tombstoning the objects the target transaction minted: `StrictInverse` conflicts (`TombstonedTarget`) if any minted object was already tombstoned; `BestEffort` tombstones the survivors; `Cascade` is `StrictInverse` over the same set (dependent-closure undo is a Phase-3 refinement, P11-C8).

vo → v1 Payload Migration

A vo envelope carries an identifier-only payload; a v1 envelope carries the value-typed payload this catalog defines. The two forms do not coexist as permanent dialects (that would double the reducer surface forever); instead the catalog ships a **one-time migration** that lifts a vo envelope to v1 using the score graph as context, applied once on read. Production code carries only v1 payloads; vo envelopes survive only as a regression corpus.

REQUIREMENT

The migration `migrate_v0_envelope(v0, context: &Score)` **MUST** be **deterministic** (two implementations migrating the same vo envelope against the same context produce byte-identical v1 envelopes) and **equivalence-preserving** (a vo envelope and its v1 migration reduce to byte-identical canonical `MaterializedState`). When a value cannot be reconstructed from the vo projection plus the context, the migration **MUST** report the envelope unmigratable rather than fabricate a value, and the bundle opens read-only.

The reference implementation (`epiphany-ops::migrate`) reconstructs the `InsertEvent` event, the `DeleteEvent` compensation, the `ChangeRegionTimeModel` model, the `SetUserSystemBreak` anchor, and the cross-cutting structure self-containedly from the vo projection; it recovers a `RespellPitch` spelling from the context (P12-K1, Section 3.3). The migration's merge gate (`epiphany-testkit::migration`) drives the inverse direction — projecting a v1 corpus to vo and migrating it back — and asserts byte-identical reduction plus a non-vacuity guard.

K₁ — Framework Slots (Phase 3)

This chapter drafted the remaining catalogue items as framework slots. Since the Phase-2 **M2** expansion (the broad-Ko groups in *epiphany-ops*), several have been *implemented* and are available under the Phase-2 profile; their per-primitive §schema joins Chapter 3 with the M2e catalogue expansion. The genuinely Phase-3 slots that remain **unavailable** are listed second: an implementation **MUST** reject an operation of one of *those* kinds (and **MUST** *not* reject the implemented kinds below). Each remaining slot is a schema-fill against the template of Chapter 2; adding one is not a fresh design.

Implemented since M2 (available under the Phase-2 profile)

Modify event; transpose Field overwrite on an event’s non-identity fields, and an order-dependent chromatic transpose (M2 Group 1). Discipline: last-writer-wins with structural-field-collision (Section 3.3).

Insert / delete / modify identified pitch Pitch-level mint, tombstone, and field overwrite within an event (M2 Group 1). Disciplines as for the event-level analogues.

Delete / update tie / slur / beam / spanner Cross-cutting tombstone and field overwrite (M2 Group 2). Disciplines: delete-wins with re-anchoring (Section 3.2) and field overwrite.

Create / delete region / staff instance / voice Structural container mint and *empty-only* delete (M2 Group 3). Disciplines: set-union creation (Section 3.4) and a delete-wins tombstone that is a precondition no-op unless the container has no live children.

Remaining framework slots (Phase 3 — unavailable, MUST reject)

Create score / canvas / staff The remaining structural mints (the document root, the canvas, and global staves) the Phase-2 slice does not exercise. Discipline: set-union creation.

Set metadata (title / composer / lyricist / copyright) Field overwrite on score metadata. Discipline: LWW advisory (Section 3.6).

Set metric grid / time signature / tempo segment Structural field overwrite on a region’s metric model. Discipline: last-writer-wins, with a structural-field-collision on concurrent differing grids.

Set layout / system- and page-break advisory The page/layout advisory companion to Section 3.6.

Non-Goal

The full 60–80-primitive catalogue is not a Phase-2 deliverable. The framework (Chapter 2) and the Ko representative set (Chapter 3) are sufficient to exercise every reduction discipline; the remaining primitives are Phase-3 schema-fill.