

---

# Epiphany

BINARY FORMAT

*A companion to the Core Specification*

---

Version 0.9.0 — The decode vector corpus is ratified (the cross-implementation decoder test is no longer deferred)

Normative for the byte layouts it defines

# Contents

---

|          |                                                                            |           |
|----------|----------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>About This Companion</b>                                                | <b>3</b>  |
| 1.1      | Relationship to the Core Specification and the Operation Catalog . . . . . | 4         |
| 1.2      | Conformance . . . . .                                                      | 4         |
| <b>2</b> | <b>Encoding Conventions</b>                                                | <b>6</b>  |
| 2.1      | The Three Encoding Regimes . . . . .                                       | 6         |
| 2.2      | Primitive Composition Rules . . . . .                                      | 7         |
| <b>3</b> | <b>Identifiers and Derivations</b>                                         | <b>9</b>  |
| 3.1      | The 128-Bit Identifier Family . . . . .                                    | 9         |
| 3.2      | TypedObjectId . . . . .                                                    | 9         |
| 3.3      | System-Derived Counters and Digest Truncations . . . . .                   | 10        |
| 3.4      | The Domain-Tag Registry . . . . .                                          | 11        |
| 3.5      | SnapshotId . . . . .                                                       | 11        |
| <b>4</b> | <b>Primitive Value Encodings</b>                                           | <b>13</b> |
| 4.1      | RationalTime . . . . .                                                     | 14        |
| <b>5</b> | <b>Graph Value Layouts</b>                                                 | <b>15</b> |
| 5.1      | The Score Envelope . . . . .                                               | 15        |
| 5.2      | Composition Rules as Normative Encoding Facts . . . . .                    | 16        |
| 5.3      | Discriminant Tables . . . . .                                              | 17        |
| 5.3.1    | C-style enums (one byte, no payload) . . . . .                             | 17        |
| 5.3.2    | Tagged unions (one byte, then the variant payload) . . . . .               | 17        |
| 5.4      | Representative Complete Layouts . . . . .                                  | 18        |
| 5.4.1    | RationalTime . . . . .                                                     | 18        |
| 5.4.2    | TimeAnchor . . . . .                                                       | 18        |
| 5.4.3    | Pitch . . . . .                                                            | 19        |
| 5.4.4    | Event . . . . .                                                            | 19        |
| 5.4.5    | Slur . . . . .                                                             | 19        |
| 5.5      | The CanonicalValue Seam . . . . .                                          | 19        |
| 5.6      | The Frozen-Layout Rule . . . . .                                           | 20        |
| <b>6</b> | <b>Operation Wire Forms</b>                                                | <b>21</b> |
| 6.1      | The Operation Envelope . . . . .                                           | 21        |
| 6.2      | OperationPayload and the Meta-Operations . . . . .                         | 22        |
| 6.3      | OperationKind: the Primitive Wire Discriminants . . . . .                  | 22        |
| 6.4      | OperationKindTag: a Separate Discriminant Space . . . . .                  | 24        |

|           |                                                       |           |
|-----------|-------------------------------------------------------|-----------|
| 6.5       | Stamps and Causal Contexts                            | 25        |
| 6.6       | Effects, Conflicts, Anomalies, and Materialized State | 25        |
| 6.6.1     | MaterializedState                                     | 26        |
| 6.7       | vo Payloads Have No Wire Form                         | 27        |
| <b>7</b>  | <b>Bundle Physical Layout</b>                         | <b>28</b> |
| 7.1       | The Fixed Header (64 bytes)                           | 28        |
| 7.2       | The Superblock Slots (256 bytes each)                 | 28        |
| 7.3       | Chunks                                                | 29        |
| 7.3.1     | ChunkRef (95 bytes)                                   | 30        |
| 7.4       | Operation-Envelope Block Payloads                     | 31        |
| 7.5       | The Manifest                                          | 31        |
| 7.6       | The Operation Index                                   | 33        |
| <b>8</b>  | <b>Extension Declaration Blobs and Edit Barriers</b>  | <b>34</b> |
| 8.1       | Framing                                               | 34        |
| 8.2       | EditBarrier                                           | 35        |
| <b>9</b>  | <b>Schema Evolution</b>                               | <b>37</b> |
| 9.1       | The Chunk-Level Gate                                  | 37        |
| 9.2       | What “Additive” Means Here                            | 37        |
| 9.3       | Migration Discipline                                  | 38        |
| 9.4       | Schema Major 1                                        | 38        |
| 9.4.1     | Where the changed fields reach                        | 39        |
| 9.4.2     | Cross-major reader behaviour                          | 39        |
| 9.4.3     | The accept-set gate                                   | 40        |
| 9.4.4     | Changed and new value layouts                         | 40        |
| 9.4.5     | Length-prefix unification (partial)                   | 40        |
| 9.4.6     | Migration from major 0                                | 40        |
| 9.5       | Schema Major 2                                        | 41        |
| 9.5.1     | Where the changed fields reach                        | 41        |
| 9.5.2     | Cross-major reader behaviour                          | 42        |
| 9.5.3     | Changed value layouts                                 | 43        |
| 9.5.4     | New leaf-type layouts                                 | 43        |
| 9.5.5     | Migration from major 1                                | 44        |
| <b>10</b> | <b>Non-Canonical Pinned Encodings</b>                 | <b>46</b> |
| 10.1      | LayoutObjectId (MUSCLOID)                             | 46        |
| 10.2      | ResolvedLayoutIR Canonical Output                     | 46        |
| 10.3      | Layout Caches and Font Metrics                        | 47        |
| <b>11</b> | <b>Golden Anchor Registry</b>                         | <b>48</b> |
| 11.1      | The Decode Vector Corpus                              | 50        |
| <b>12</b> | <b>Revision History</b>                               | <b>51</b> |

# About This Companion

---

The *Binary Format* document is a companion to the Epiphany Core Specification. It fulfils the core specification’s delegation in Chapter 8, BINARY FORMAT COMPANION (the `sec:format:binary` section): the byte-level encoding — integer conventions, exact field widths, record alignment, endianness, string encoding details, struct layouts — is delivered here. In the core specification’s words: “*The Binary Format document is normative; an implementation cannot conform to the file format specification without conforming to the Binary Format specification.*”

This release (v0.1.0) delivers the full **K-independent** encoding surface — identifier and derivation layouts, primitive value encodings, the whole-`Score` composite value codec, the operation-layer wire forms, the bundle physical layout, and the extension-declaration blob forms — *plus* the **Ko/M2 operation-payload wire framing**. The Operation Catalog’s Ko set (the representative primitives and the M2 broad-Ko groups, catalog version 0.4.0) now exists in ratified form, so the historical “waits for Agent K” clause in the Phase-2 charter is discharged: every operation kind the catalog defines has its literal wire form pinned in Chapter 6.

This document does *not* cover:

- the canonical s-expression form — that is the *Text Projection* companion’s (version 0.1.0, which now exists);
- per-profile feature lists — the *Profile Conformance* companion’s;
- the Chapter-4 tuning-catalog values (pitch-space and tuning-system registries), which are undelivered Track-C work and have no wire form yet;
- the wire-format fuzzer, an *implementation* deliverable tracked in the Phase-2 process documents rather than part of this document’s normative text; Chapter 11 records the golden anchors any such harness must reproduce.

It *does* cover the cross-implementation decoder test, which is no longer deferred: Section 11.1 ratifies the literal-byte vector corpus and the accept/reject verdict a conforming decoder owes it.

## 1.1 Relationship to the Core Specification and the Operation Catalog

This companion *inherits* — it never re-derives — the core specification’s ratified convention baseline: core specification Chapter 8, BINARY FORMAT COMPANION, requirement `req:format:codec-conventions`, together with the CANONICAL BYTE-LAYOUT REFERENCE appendix (`app:bytes`). That appendix consolidates every discriminant table, derivation preimage, and primitive encoding Pass 11 and Pass 12 ratified, precisely so that this document could import it as a starting point rather than recovering the layouts from the crates. Chapter 3 therefore reproduces only *compact* normative tables, each with a citation back to the appendix row and requirement that governs it; where this document and a ratified core requirement disagree, **the core requirement governs** and the discrepancy is a defect in this document.

What this document adds is the completion of the appendix’s deferred list (core specification, LAYOUTS DEFERRED TO THE COMPANIONS, `sec:bytes:deferred`): the `OperationKind` and `OperationKindTag` discriminants and full per-operation payload encodings; the full composite struct layouts (whole-Score, operation envelopes, the manifest body’s field order); schema-version wire evolution; and the varint question, which Chapter 2 settles.

The division of labour with the *Operation Catalog* is exact: the catalog fixes each operation’s **field set and order** (Operation Catalog, PER-PRIMITIVE SCHEMA TEMPLATE) and its semantics; this document pins the **bytes** of those fields. Chapter 6 cites the catalog section for every payload rather than restating what the fields mean. Likewise, graph *types* are the core specification’s Chapters 2–5, and the structural layer of the bundle — what a header, superblock, chunk graph, or manifest *is* and how it behaves — is core Chapter 8; this document specifies only how those structures’ bits are laid out.

### RATIONALE

**Versioning.** This companion is versioned independently of the core specification (independent semver), like the Operation Catalog — but tied tighter to it: the binary-format MAJOR version tracks the core specification’s major version (both are pre-1.0 working drafts today, hence v0.1.0), while MINOR versions diverge freely. Byte layouts change on a slower, more deliberate cadence than prose; a core-spec minor revision that touches no wire fact requires no revision here, and an additive discriminant append here (Chapter 9) requires no core-spec revision.

## 1.2 Conformance

The byte layouts in this document are **normative**. An implementation conforms to the Epiphany binary format if and only if, for every layout this document defines, it produces exactly the specified bytes when encoding and accepts exactly the specified language of byte strings when decoding.

Decode discipline is **reject, never normalize** — the workspace-wide convention the core specification establishes in Appendix D (canonical serialization determinism and ordered iteration). Concretely, each of the following is a *decode error*, never an input to be repaired:

- an unknown discriminant in any tagged union;
- a set- or map-valued field whose elements are not in strictly ascending canonical order, or which contains duplicates;
- text that is not in Unicode NFC where NFC is required (catalog identifiers, barrier pitch-space names, operation labels);
- trailing bytes after a complete top-level value (Requirement 2.2);
- recursion beyond a normative depth bound (Requirement 8.3).

A decoder **MUST NOT** silently sort, deduplicate, NFC-fold, truncate, or otherwise canonicalize non-canonical input: two implementations that disagree on repair would disagree on canonical state.

The **golden anchor registry** (Chapter 11) is the conformance contract binding this text to the reference implementation: every layout in this document is anchored by at least one named test in the reference crates, and a change that breaks an anchor is either a defect or a deliberate, versioned format revision (Chapter 9).

# Encoding Conventions

This chapter restates the ratified convention baseline (core specification, requirement `req:format:codec-conventions`) as normative fact and completes it: the three prefix/endianness regimes, the primitive composition rules, the varint disposition, and the trailing-bytes rule.

## 2.1 The Three Encoding Regimes

Three internally consistent regimes coexist in the format. Every layout in this document belongs to exactly one of them, and each layout’s chapter says which.

| Regime | Surfaces                                                                                                                                                                                                                      | Rules                                                                                                                                                                                                                                                                                  |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (a)    | Canonical document encodings: the core value codec (Chapter 5), the operation layer (Chapter 6), the bundle codec (Chapter 7).                                                                                                | Little-endian fixed-width integers; <code>u32</code> little-endian counts and length prefixes on every variable-width field.                                                                                                                                                           |
| (b)    | Layout-side pinned encodings: the extension-declaration blobs and edit-barrier trees (Chapter 8) and the <code>ResolvedLayoutIR</code> canonical output (Chapter 10).                                                         | Little-endian integers; <code>u64</code> little-endian counts and length prefixes — a golden-locked divergence from regime (a). Schema major 1 moves the <i>resolved-layout</i> surface to <code>u32</code> ; the manifest-embedded barrier blobs stay <code>u64</code> (Section 9.4). |
| (c)    | Identifiers and hash-derived forms: the 128-bit identifier family, <code>ReplicaId</code> , <code>OperationId</code> , <code>ConflictId</code> , the <code>TypedObjectId</code> discriminant, digest truncations (Chapter 3). | Big-endian fixed width. Numeric order equals canonical byte order, which is what Appendix D’s ordered iteration requires.                                                                                                                                                              |

RATIONALE

In schema major 0, regimes (a) and (b) differ only in prefix width, and the divergence was *tolerated rather than repaired*: the two surfaces were golden-locked independently (the barrier blobs and the resolved-layout output carry their own literal-byte anchors), they

never embed one another’s framing, and unifying the prefix width within major 0 would break locked bytes for zero benefit. Once a major bump was scheduled for data-model reasons, part of the unification became free to carry: schema major 1 moves the resolved-layout surface (an independent non-canonical `LayoutCache`) to regime (a)’s `u32` prefixes and re-locks its goldens. The barrier blobs cannot follow in the same step — they are carried opaquely inside the *canonical* manifest, which stays major 0 — so they keep regime (b) until a manifest-revising major (Chapter 9). Regime (c) is not a divergence at all: an identifier’s byte form *is* its sort key, so big-endian is load-bearing.

## 2.2 Primitive Composition Rules

The following rules apply throughout regime (a); regimes (b) and (c) state their deltas in their own chapters.

**Booleans.** A single byte: 0 = false, 1 = true. Any other value is a decode error.

**Integers.** Fixed-width little-endian two’s complement (`u8/i8` through `u128`), except within regime (c).

**Option<T>.** A presence byte: 0 = absent (no further bytes), 1 = present, followed by `T`’s encoding. A presence byte greater than 1 is a decode error.

**Sequences, sets, maps.** A `u32` little-endian element count, then each element (for maps: key then value). Set- and map-valued fields are emitted in canonical iteration order — ascending by the key’s canonical byte form — and decoders reject out-of-order or duplicated elements wherever the field is a set or map (as opposed to an order-significant list).

**Free text.** A `u32` little-endian byte length, then the raw UTF-8 bytes. Free-text fields (e.g. a score title) are **never** NFC-folded by the codec:  $\text{decode}(\text{encode}(x)) = x$  is the round-trip identity, and folding would break it. By contrast, *catalog identifiers* (pitch-space ids, tuning-system ids, and the other `catalog_id` newtypes) are NFC-normalized *at construction*, so their stored form is already NFC; the codec encodes the stored string verbatim. Operation-layer text fields (the transaction label, conflict field paths) are NFC-normalized at encode time (Chapter 6).

**Structs.** Positional, unframed concatenation of the fields’ encodings in declaration order. No field tags, no per-field lengths, no padding, no alignment. (This is the layout the frozen-layout rule, Requirement 5.3, freezes.)

**Tagged unions.** A single discriminant byte, then the selected variant’s payload encoded positionally. Two documented exceptions: `TypedObjectId` uses a 16-bit *big-endian* discriminant (core requirement `req:graph:typed-object-id-discriminants`; Chapter 3), and the bundle’s `ProfileId` uses a `u32` little-endian discriminant (core requirement `req:format:profileid-discriminants`; Chapter 7).

**Determinism-canonical leaves.** Where a composite codec embeds a value whose canonical byte form is owned by the determinism layer or the identifier layer (a graph identifier, `ContentHash`, `CanonicalF64`, `RationalTime`, a wall-clock integer), the core value codec frames it as a `u32` little-endian length prefix followed by the leaf’s own canonical bytes

— even when the leaf is fixed-width. The leaf’s bytes are then self-describing under its own decoder. (The operation layer instead embeds fixed-width leaves *raw*, unprefixes; Chapter 6 states this explicitly.)

**REQUIREMENT 2.1**

**No varint.** Schema major 0 defines *no* variable-length integer encoding. Every integer field in every layout in this document is fixed-width. The core specification’s “varint conventions” language (Chapter 8, BINARY FORMAT COMPANION) is hereby discharged as *none*: an encoder **MUST NOT** emit, and a decoder **MUST NOT** accept, any LEB128-style or otherwise variable-width integer under schema major 0. Variable width in this format comes only from explicit `u32/u64` length prefixes and from `RationalTime`’s length-prefixed big-integer magnitudes (Chapter 4).

**REQUIREMENT 2.2**

**Trailing bytes.** Every top-level decode entry point — a whole-`Score` value, a per-value `CanonicalValue` decode, an `OperationKindTag`, a `MaterializedState`, a manifest payload, an operation-index payload, a block payload, an extension-declaration blob, a fixed-width primitive — **MUST** reject input with bytes remaining after the complete value has been decoded. Trailing bytes are a decode error, never ignored padding.

# Identifiers and Derivations

This chapter is an *import-by-citation* of the core specification’s CANONICAL BYTE-LAYOUT REFERENCE appendix (`app:bytes`): that appendix and its golden anchors govern every table in this chapter, and the tables here are compact restatements for self-containedness, not a second source of truth. All layouts in this chapter are regime (c): big-endian, fixed width.

## 3.1 The 128-Bit Identifier Family

**ReplicaId.** 8 bytes, big-endian u64. The reserved system namespace is `ReplicaId::SYSTEM_DERIVED = 0xFFFF_FFFF_FFFF_FFFF`; user-authored replicas **MUST NOT** use it.

**Typed graph identifiers.** Every typed 128-bit graph identifier (`EventId`, `PitchId`, `VoiceId`, `RegionId`, `StaffInstanceId`, `TransactionId`, ... — the `graph_id` family) packs the 64-bit replica into the high half and the counter into the low half, and encodes as exactly 16 bytes: replica big-endian (8) || counter big-endian (8). Numeric order, canonical byte order, and Appendix-D iteration order coincide.

**OperationId.** The same 16-byte form: replica big-endian (8) || counter big-endian (8). Ordering is (*replica*, *counter*) lexicographic, which equals byte order.

**Registry and author identifiers (operation layer).** The opaque 128-bit registry newtypes (`OperationKindRegistryId`, `ConflictKindRegistryId`, `ResolutionRegistryId`, `RepairKindRegistryId`, `ReanchorReasonRegistryId`, `ReplicaAnomalyRegistryId`, `IntegrityAnomalyRegistryId`, `ExtensionPreconditionId`, `PreconditionFailureRegistryId`) and `AuthorId` all encode as 16 big-endian bytes.

## 3.2 TypedObjectId

Canonical form: a **16-bit big-endian discriminant** || the variant payload (core requirement `req:graph:typed-object-id-discriminants`). Non-Registered variants carry one 16-byte identifier, for a total of **18 bytes**; Registered carries the `ObjectKindRegistryId` (16 big-endian bytes) || the extension’s own raw u128 (16 big-endian bytes), for a total of **34 bytes**. An unknown discriminant is a decode error.

| Disc | Variant | Disc | Variant |
|------|---------|------|---------|
| 0    | Event   | 14   | Spanner |
| 1    | Pitch   | 15   | Marker  |

| Disc | Variant               | Disc | Variant              |
|------|-----------------------|------|----------------------|
| 2    | Voice                 | 16   | AnalyticalAnnotation |
| 3    | Staff                 | 17   | Comment              |
| 4    | StaffInstance         | 18   | GraphicObject        |
| 5    | StaffGroup            | 19   | GraphicGesture       |
| 6    | Region                | 20   | TimeSignature        |
| 7    | Instrument            | 21   | AnalysisLayer        |
| 8    | PartDefinition        | 22   | Tuplet               |
| 9    | Measure               | 23   | RepeatStructure      |
| 10   | BarlineAlignmentGroup | 24   | LyricLine            |
| 11   | Slur                  | 25   | ChordSymbol          |
| 12   | Tie                   | 26   | View                 |
| 13   | Beam                  | 27   | Registered           |

### 3.3 System-Derived Counters and Digest Truncations

All content derivation uses BLAKE3-256 over a domain-separated preimage: the 8-byte domain tag is always the first bytes hashed. Two truncations recur, both taking the digest’s *leading* bytes big-endian:

- $\text{trunc64}(d) = \text{u64} :: \text{from\_be\_bytes}(d[0..8])$  — the counter of a system-derived identifier;
- $\text{trunc128}(d) = \text{u128} :: \text{from\_be\_bytes}(d[0..16])$  — content-derived 128-bit identifiers.

The system-derived counter function is  $\text{trunc64}(\text{BLAKE3}(\text{tag} \parallel \text{inputs}))$ ; the resulting identifier is  $\text{from\_parts}(\text{ReplicaId}::\text{SYSTEM\_DERIVED}, \text{counter})$ . Only a system domain tag (prefix MUSCS) may seed it. The ratified instances (governed by `app:bytes` and the cited requirements):

| Identifier         | Tag      | Canonical inputs                                                                                                                                          |
|--------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Promoted VoiceId   | MUSCSVCE | 64-byte preimage: <code>staff_instance (16 BE)    original_voice (16 BE)    winning_op (16 BE)    losing_op (16 BE)</code> . Core SYSTEM-PROMOTED VOICES. |
| System PitchId     | MUSCSPCH | The pitch’s intrinsic canonical bytes (core requirement <code>req:graph:system-derived-pitch-id</code> ).                                                 |
| IntegrityAnomalyId | MUSCSANM | The anomaly kind’s canonical bytes (Chapter 6; core requirement <code>req:graph:integrity-anomaly-id</code> ).                                            |

The 128-bit content-derived identifiers:

| Identifier | Tag      | Preimage after the tag                                                                                                                                                                                                          |
|------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ConflictId | MUSCCONF | <code>kind.canonical_bytes()    causing operations (each 16 BE, sorted by canonical bytes)    affected objects (each TypedObjectId canonical bytes, sorted); trunc128</code> . Wire form of the id itself: 16 big-endian bytes. |

| Identifier     | Tag      | Preimage after the tag                                                                                                                                                                                                                                        |
|----------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ManifestId     | MUSCMNIF | document_id (16)    generation (u64 LE)    manifest body bytes (body excludes the manifest_id field); trunc128. Restated normatively in Chapter 7 (core requirement req:format:manifest-id). Note the <i>on-disk</i> field encodes little-endian (Chapter 7). |
| EnvelopeHash   | MUSCENVH | The envelope’s complete canonical bytes; the full 32-byte digest (no truncation). Chapter 6.                                                                                                                                                                  |
| LayoutObjectId | MUSCLOID | Non-canonical; three preimage shapes, Chapter 10 (core requirement req:layoutir:object-id-derivation).                                                                                                                                                        |
| BlobId         | MUSCBLOB | The blob payload, bare (req:format:blob-hash-shape); full 32-byte digest. Chapter 4.                                                                                                                                                                          |

### 3.4 The Domain-Tag Registry

Every domain tag is exactly 8 printable-ASCII bytes beginning `MUSC`; the vocabulary is closed (governed by `app:bytes`, `DOMAIN-TAG REGISTRY`). Extension system tags **MUST** begin `MUSCS`, be exactly 8 bytes, and collide with neither the built-ins nor the file magics.

| Tag       | Name             | Use                                                       |
|-----------|------------------|-----------------------------------------------------------|
| MUSCCHNK  | chunk            | non-manifest chunk hash preimages                         |
| MUSCMANI  | manifest         | manifest chunk hash preimages                             |
| MUSCBLOB  | blob             | BlobId (bare tag    payload)                              |
| MUSCCONF  | conflict         | ConflictId derivation                                     |
| MUSCENVH  | envelope         | EnvelopeHash                                              |
| MUSCFNTM  | font metrics     | glyph-catalog metrics hash (non-canonical)                |
| MUSCMNIF  | manifest id      | ManifestId derivation                                     |
| MUSCSVCE  | system voice     | promoted-voice counters                                   |
| MUSCSPCH  | system pitch     | system-derived pitch counters                             |
| MUSCSANM  | system anomaly   | IntegrityAnomalyId                                        |
| MUSCLOID  | layout object id | LayoutObjectId (non-canonical)                            |
| MUSCBND\0 | bundle magic     | file magic, <i>not</i> a hash tag (note the trailing NUL) |
| MUSCSUPR  | superblock magic | slot magic, <i>not</i> a hash tag                         |

### 3.5 SnapshotId

The core specification’s support-type identity table says “*the Binary Format companion defines the exact derivation*” of `SnapshotId`. `vo` has no snapshot producer — pruning and canonical-base creation are deferred — so there is nothing to derive *from* yet.

#### REQUIREMENT 3.1

In schema major 0, `SnapshotId` is an **opaque 16-byte identifier**: 16 raw bytes on the wire (Chapter 4), with no normative derivation function. Readers **MUST** treat two `SnapshotIds` as equal iff their 16 bytes are equal and **MUST NOT** attempt to derive or verify a snapshot id from snapshot content.

**OPEN QUESTION**

The `SnapshotId` derivation (presumably a content derivation in the `trunc128` family, over the snapshot's canonical payload and its covering frontier) is deferred together with the pruning/garbage-collection track that will produce the first snapshots. It must be pinned before any writer emits a `canonical_base`; until then the opaque reading above is complete.

# Primitive Value Encodings

The leaf encodings owned by the determinism layer and the small fixed identifier types. These byte forms are what the framings of Chapters 5–7 embed.

| Type                                                                        | Width | Encoding                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CanonicalF64                                                                | 8     | IEEE-754 binary64, little-endian. $-0.0$ is canonicalized to $+0.0$ at construction, so the negative-zero bit pattern never appears in canonical bytes. Decode rejects NaN and $\pm\infty$ (non-finite floats are corruption).                                                              |
| QuantizedCoord                                                              | 8     | i64 little-endian, in units of $1/1024$ staff space ( <code>STAFF_SPACE_GRID = 1024</code> ). Construction from a float rounds ties-to-even and rejects non-finite or out-of-range input.                                                                                                   |
| ContentHash                                                                 | 32    | Raw BLAKE3-256 digest bytes.                                                                                                                                                                                                                                                                |
| ChunkId                                                                     | 32    | Raw digest bytes (a <code>ContentHash</code> newtype).                                                                                                                                                                                                                                      |
| DomainTag                                                                   | 8     | The raw ASCII tag bytes. Decode validates against the closed vocabulary of Section 3.4; foreign bytes are a decode error.                                                                                                                                                                   |
| WallClockTime                                                               | 8     | i64 little-endian, nanoseconds. Floating-point wall-clock time is forbidden in stored data.                                                                                                                                                                                                 |
| WallClockDuration                                                           | 8     | i64 little-endian, nanoseconds.                                                                                                                                                                                                                                                             |
| SchemaVersion                                                               | 4     | major u16 LE    minor u16 LE.                                                                                                                                                                                                                                                               |
| SemVer                                                                      | 12    | major u32 LE    minor u32 LE    patch u32 LE. Canonical ordering of <code>SemVer</code> -keyed collections is <b>numeric</b> on ( <i>major, minor, patch</i> ), never byte-lexicographic on the little-endian encoding (which would order <code>256.0.0</code> before <code>1.0.0</code> ). |
| Reduction                                                                   | 4     | u32 little-endian.                                                                                                                                                                                                                                                                          |
| AlgorithmVersion                                                            | 4     | u32 little-endian.                                                                                                                                                                                                                                                                          |
| FileUuid, DocumentId, LineageId, SnapshotId, ExtensionId, ProfileRegistryId | 16    | 16 raw opaque bytes each. No internal structure is interpreted; equality is byte equality.                                                                                                                                                                                                  |
| BlobId                                                                      | 32    | Raw digest bytes: <code>BLAKE3(MUSCBLOB    payload)</code> , the only bare <i>tag    payload</i> content hash in the format (core requirement <code>req:format:blob-hash-shape</code> ).                                                                                                    |
| FrontierBytes                                                               | var   | u32 LE length    opaque bytes (an ops-computed causal frontier the bundle carries verbatim).                                                                                                                                                                                                |

## 4.1 RationalTime

Restating core requirement `req:format:rationaltime-encoding`, which governs. `RationalTime` is an arbitrary-precision rational, always stored *reduced* (equal rationals encode identically; the sign lives on the numerator; the denominator is strictly positive). Its canonical byte form:

| Field       | Encoding                                                   |
|-------------|------------------------------------------------------------|
| sign        | 1 byte: 0 = zero, 1 = positive, 2 = negative               |
| numerator   | u32 LE magnitude byte length    big-endian magnitude bytes |
| denominator | u32 LE magnitude byte length    big-endian magnitude bytes |

`Decode` rejects a sign byte greater than 2, a zero denominator, and trailing bytes. `MusicalPosition` and `MusicalDuration` are newtypes whose canonical bytes are exactly their inner `RationalTime`'s. Wall-clock integers deliberately do *not* use this form: they are the fixed-width little-endian integers of the table above, matching `QuantizedCoord`'s convention.

# Graph Value Layouts

---

This chapter defines the schema-major-0 wire form of every value reachable from a `Score` — the whole-document composite codec. The core specification’s `DECISIONS` record (P11-4) held this codec *provisional* pending this companion; Requirement 5.1 and Requirement 5.3 **ratify** it: the reference implementation’s layout, as specified here, *is* the schema-major-0 wire form, and P11-4’s provisional status is discharged.

All layouts in this chapter are regime (a): little-endian, `u32` prefixes.

## 5.1 The Score Envelope

### REQUIREMENT 5.1

The canonical bytes of a `Score` are the bare positional concatenation of its nineteen fields’ encodings, in exactly this order:

1. `metadata`
2. `canvas`
3. `instruments`
4. `staves`
5. `staff_groups`
6. `parts`
7. `cross_cutting`
8. `time_signatures`
9. `tuning_context`
10. `tempo_map`
11. `events`
12. `spelling_attachments`
13. `decomposition_attachments`
14. `spelling_precedence`
15. `analysis_layers`
16. `views`

- 17. identity
- 18. tombstoned\_pitches
- 19. tombstoned\_events

There is **no magic** and **no version prefix** at the value level: format identification lives in the bundle’s fixed header (Chapter 7) and versioning lives at the chunk layer (SchemaVersion in every ChunkRef and superbloc; Chapter 9). Decoders **MUST** reject trailing bytes after the nineteenth field.

## 5.2 Composition Rules as Normative Encoding Facts

The reference codec is built from a small set of conventions; each is a normative statement about the bytes, not an implementation detail:

**Positional structs.** Every struct in this chapter encodes as its fields in declaration order, unframed (Section 2.2). Zero-field structs encode as zero bytes.

**C-style enums.** A fieldless enum is a single discriminant byte (tables in Section 5.3).

**Tagged unions.** A discriminant byte, then the variant’s fields positionally.

**Leaf framing.** The determinism-canonical leaf types — ReplicaId, OperationId, every embedded typed graph identifier (EventId, PitchId, VoiceId, StaffId, StaffInstanceId, StaffGroupId, RegionId, InstrumentId, PartDefinitionId, MeasureId, BarlineAlignmentGroupId, SlurId, TieId, BeamId, SpannerId, TupletId, MarkerId, AnalyticalAnnotationId, CommentId, RepeatStructureId, LyricLineId, ChordSymbolId, GraphicObjectId, GraphicGestureId, TimeSignatureId, AnalysisLayerId, ViewId) — plus ContentHash, CanonicalF64, RationalTime, MusicalPosition, MusicalDuration, WallClockTime, and WallClockDuration, are each emitted as u32 LE length || the leaf’s canonical bytes (Chapters 3 and 4). An embedded 16-byte identifier therefore occupies 20 bytes inside a Score. Decode runs the leaf’s own validating decoder over the framed region.

**Catalog identifiers.** The catalog\_id newtypes (PitchSpaceId, TuningSystemId, AccidentalId, registry-name ids, ...) encode as their stored NFC string under the String rule (u32 LE length || UTF-8).

**EventArena.** u32 LE event count, then each Event in ascending-EventId order (identity travels inside each event; the arena adds no per-event framing).

**Validated reconstruction.** Types with checked constructors decode through them: TupletRatio rejects degenerate ratios, KeySignature encodes its fifths() as i8 and rejects out-of-range values, TimeSignature re-validates that beat groups sum to the measure duration, NonZeroU16 rejects zero, EventOrderingDAG rejects cyclic edge maps, ReferencePitch, Tempo, and SpellingPrecedence re-validate their invariants. Structurally well-formed bytes that fail a type invariant are a decode error (reject-never-normalize).

## 5.3 Discriminant Tables

The complete discriminant assignment for schema major 0. Every table is governed by Requirement 5.3: an unknown discriminant is a decode error; changes are gated by Chapter 9.

### 5.3.1 C-style enums (one byte, no payload)

| Type                    | Discriminants                                                                    |
|-------------------------|----------------------------------------------------------------------------------|
| MeasurePosition         | 0 Start, 1 End                                                                   |
| RegionEdge              | 0 Start, 1 End                                                                   |
| CmnNominal              | 0 C, 1 D, 2 E, 3 F, 4 G, 5 A, 6 B                                                |
| SpellingSourceKind      | 0 UserChosen, 1 Imported, 2 Propagated, 3 Inferred, 4 Analytical                 |
| TempoShape              | 0 Constant, 1 Linear, 2 Exponential, 3 Curve                                     |
| StemDirection           | 0 Up, 1 Down                                                                     |
| MeasureNumberVisibility | 0 Auto, 1 Always, 2 Never                                                        |
| Aleatoric               | 0 Musical, 1 WallClock, 2 EitherPerEvent, 3 FreelyMixed                          |
| AnchoringDiscipline     |                                                                                  |
| NoteValue               | 0 Whole, 1 Half, 2 Quarter, 3 Eighth, 4 Sixteenth, 5 ThirtySecond, 6 SixtyFourth |
| ClefShape               | 0 G, 1 F, 2 C, 3 Percussion                                                      |

### 5.3.2 Tagged unions (one byte, then the variant payload)

| Type                | Discriminants                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------|
| AnchorOffset        | 0 Musical, 1 WallClock, 2 Zero                                                                |
| TimeAnchor          | 0 Event, 1 Measure, 2 Region, 3 WallClock (Section 5.4)                                       |
| EventPosition       | 0 Musical, 1 WallClock                                                                        |
| ConcreteDuration    | 0 Musical, 1 WallClock                                                                        |
| EventDuration       | 0 Musical, 1 WallClock, 2 Indeterminate                                                       |
| TimeBounds          | 0 MusicalRange, 1 WallClockRange, 2 Unbounded                                                 |
| PitchSpacePosition  | 0 Cmn, 1 Integer, 2 JiVector, 3 Registered                                                    |
| TuningReference     | 0 Inherit, 1 Explicit                                                                         |
| AcousticRealization | 0 Implicit, 1 CentsOffset, 2 AbsoluteHz                                                       |
| SpellingNominal     | 0 Cmn, 1 Integer, 2 Registered                                                                |
| SpellingSource      | 0 UserChosen, 1 Inferred, 2 Imported, 3 Propagated, 4 Analytical                              |
| SpellingScope       | 0 Pitch, 1 Range                                                                              |
| SpellingDirective   | 0 Explicit, 1 Rule                                                                            |
| VoiceSelector       | 0 All, 1 Voices                                                                               |
| GraceKind           | 0 Acciaccatura, 1 Appoggiatura, 2 Unmeasured, 3 Measured-Fraction                             |
| IndeterminacyKind   | 0 Pitch, 1 Duration, 2 Choice, 3 Compound                                                     |
| TrajectoryEndpoint  | 0 EventPitch, 1 ExplicitPitch                                                                 |
| TrajectoryShape     | 0 Linear, 1 Exponential, 2 Curve, 3 Stepwise                                                  |
| Event               | 0 Pitched, 1 Unpitched, 2 Rest, 3 Indeterminate, 4 Trajectory, 5 Graphic, 6 Cue (Section 5.4) |
| RegionTimeModel     | 0 Metric, 1 Proportional, 2 Aleatoric                                                         |
| RegionContent       | 0 StaffBased, 1 FreeGraphic, 2 Hybrid                                                         |

| Type                 | Discriminants                                                                 |
|----------------------|-------------------------------------------------------------------------------|
| VoiceOrigin          | 0 UserDeclared, 1 Imported, 2 SystemPromoted                                  |
| StaffGroupKind       | 0 GrandStaff, 1 Bracket, 2 SubBracket, 3 Choral, 4 Registered                 |
| TieClass             | 0 Standard, 1 Editorial, 2 CrossVoice, 3 LaissezVibrer, 4 Registered          |
| AnnotationAnchor     | 0 Event, 1 Range, 2 Region                                                    |
| GestureAnchoring     | 0 Events, 1 Range, 2 Free                                                     |
| DecompositionSource  | 0 UserChosen, 1 Inferred, 2 Imported, 3 Propagated                            |
| TimeSignatureDisplay | 0 Standard, 1 Compound, 2 Irrational, 3 MixedDenominators, 4 None, 5 Symbolic |

**REQUIREMENT 5.2**

**The `SpellingSource` / `SpellingSourceKind` trap.** These two vocabularies name the same five provenance concepts but assign them *different* discriminants: `SpellingSourceKind` (the C-style precedence-list element) is 0 UserChosen, 1 Imported, 2 Propagated, 3 Inferred, 4 Analytical, while `SpellingSource` (the payload-carrying attachment provenance) is 0 UserChosen, 1 Inferred, 2 Imported, 3 Propagated, 4 Analytical. The orders differ (`Inferred` is 3 in one and 1 in the other), both are golden-locked, and an implementation **MUST NOT** share one discriminant function between them.

## 5.4 Representative Complete Layouts

The following layouts are spelled byte-by-byte both as worked examples of the composition rules and as normative fact. Recall that inside this chapter’s codec every leaf is framed (`u32 LE length || bytes`), so an embedded 16-byte identifier occupies  $4 + 16 = 20$  bytes.

### 5.4.1 RationalTime

As a leaf inside a composite: `u32 LE length || the canonical form of Section 4.1 (sign || length-prefixed big-endian numerator magnitude || length-prefixed big-endian denominator magnitude)`.

### 5.4.2 TimeAnchor

One discriminant byte, then:

**0 Event** `id (EventId leaf, 20) || offset (AnchorOffset: tag byte, then for Musical a MusicalDuration leaf, for WallClock a WallClockDuration leaf ( $4 + 8 = 12$ ), for Zero nothing)`.

**1 Measure** `id (MeasureId leaf, 20) || position (MeasurePosition, 1) || offset (AnchorOffset)`.

**2 Region** `id (RegionId leaf, 20) || edge (RegionEdge, 1) || offset (AnchorOffset)`.

**3 WallClock** `time (WallClockTime leaf, 12)`.

### 5.4.3 Pitch

Positional struct: `scale_position || acoustic`.

- `ScalePosition` = `space` (`PitchSpaceId`: NFC string, u32 LE length || UTF-8) || `position` (`PitchSpacePosition`: tag byte, then `Cmn` = nominal (`CmnNominal`, 1) || `alteration` (i8, 1) || `octave` (i8, 1); `Integer` = `space_size` (u16 LE, 2) || `index` (i32 LE, 4); `JiVector` = u32 LE count || that many i32 LE components; `Registered` = a catalog-id string).
- `AcousticPitch` = `tuning` (`TuningReference`: tag byte; Explicit carries a `TuningSystemId` catalog-id string) || `realization` (`AcousticRealization`: tag byte; `CentsOffset` and `AbsoluteHz` each carry a `CanonicalF64` leaf,  $4 + 8 = 12$ ).

### 5.4.4 Event

One discriminant byte (0–6), then the variant struct positionally. All seven variants share the prefix `id` (`EventId` leaf) || `voice` (`VoiceId` leaf) || `position` (`EventPosition`) || `duration` (`EventDuration`); the remaining fields, in order:

| Disc | Variant       | Fields after the common prefix                                                                                                                                 |
|------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0    | Pitched       | <code>pitches</code>    <code>articulations</code>    <code>dynamic</code>    <code>ornaments</code>    <code>stem</code>    <code>grace</code>                |
| 1    | Unpitched     | <code>staff_position</code>    <code>instrument_member</code>    <code>articulations</code>    <code>dynamic</code>    <code>stem</code>    <code>grace</code> |
| 2    | Rest          | <code>vertical_position</code>    <code>visible</code>                                                                                                         |
| 3    | Indeterminate | <code>indeterminacy</code>    <code>hints</code>                                                                                                               |
| 4    | Trajectory    | <code>start</code>    <code>end</code>    <code>shape</code>    <code>display</code>                                                                           |
| 5    | Graphic       | <code>graphics</code>    <code>playback_bindings</code>                                                                                                        |
| 6    | Cue           | <code>source</code>    <code>rendering</code>                                                                                                                  |

Each field encodes under this chapter’s rules for its type (vectors are u32-counted, options carry a presence byte, embedded identifiers are leaves, unions carry their tag byte).

### 5.4.5 Slur

Positional struct of three leaves: `id` (`SlurId` leaf, 20) || `start_event` (`EventId` leaf, 20) || `end_event` (`EventId` leaf, 20) — 60 bytes total.

## 5.5 The CanonicalValue Seam

The public per-value codec — the seam between this document and the Operation Catalog’s value-typed payloads (Operation Catalog, VALUE-TYPED PAYLOADS, requirement `req:catalog:value-encoding`) — is defined for exactly these twenty-three types (the Phase-3 operation tranche added `Staff/TimeSignature/TempoSegment/StaffLineConfiguration`; the repeat-authoring revision added `RepeatStructure`):

`Event`, `Rest`, `Pitch`, `IdentifiedPitch`, `PitchSpelling`, `Tie`, `Slur`, `Beam`, `Spanner`, `RegionTimeModel`, `TimeAnchor`, `Region`, `StaffInstance`, `Voice`,

`DecompositionAttachment, SpellingSourceKind, ScoreMetadata, MetricGrid, Staff, TimeSignature, TempoSegment, StaffLineConfiguration, RepeatStructure.`

The seam introduces **no new bytes**: a value’s stand-alone canonical bytes are byte-for-byte the bytes the whole-`Score` codec embeds for that value. Stand-alone decode applies the same validation and rejects trailing bytes.

## 5.6 The Frozen-Layout Rule

### REQUIREMENT 5.3

**Frozen positional layouts (the schema-evolution keystone).** Within schema major 0, every positional struct layout in this chapter is **frozen**: there is no field-addition mechanism at the value level — no per-field framing, no optional-field encoding, no trailing-field tolerance. Any change to any struct’s field *set* or field *order* is a schema-MAJOR change and requires a documented migration in this document’s revision history (Chapter 9). Appending a *new variant* to an **open** discriminant vocabulary is a schema-MINOR change under the rules of Chapter 9. At the value layer, the open vocabularies are those with a Registered **escape variant** — `PitchSpacePosition`, `SpellingNominal`, `StaffGroupKind`, `TieClass` — through which extensions attach without any wire change at all; every *other* union in Section 5.3 (including `Event`, `TimeAnchor`, `TimeSignatureDisplay`) is closed in vo and may grow only by an append ratified in a revision of this document. This rule ratifies the project’s staging decision that data-model payload expansion — `SlurKind`, beam geometry, voltas, instrument bodies, score-metadata growth, and their kin — lands as a coordinated schema-MAJOR revision of this document, not as ad-hoc field insertion.

# Operation Wire Forms

This chapter pins the operation layer: envelopes, payloads, stamps, causal contexts, effects, and materialized state. It **ratifies** the encodings the ops crate’s `DECISIONS` record held as “provisional canonical encoding” (mirroring core P11-4); that provisional status is discharged.

Regime (a) applies, with one composition difference from Chapter 5: the operation layer embeds *fixed-width* canonical leaves (identifiers, hashes, `ConflictId`) **raw**, without a length prefix (`push_canon`); only variable-width parts carry `u32` LE length prefixes (`push_lp_bytes`). Sequences (`push_seq`) are a `u32` LE count, then each element individually length-prefixed (`u32` LE), regardless of element width. Text (`push_str`) is NFC-normalized at encode time, then length-prefixed UTF-8. Embedded graph values are the `CanonicalValue` bytes of Section 5.5, framed by a `u32` LE length prefix (Operation Catalog requirement `req:catalog:value-encoding`).

## 6.1 The Operation Envelope

### REQUIREMENT 6.1

An `OperationEnvelope`’s canonical bytes are, in order:

| Field                       | Encoding                                                                                    |
|-----------------------------|---------------------------------------------------------------------------------------------|
| <code>id</code>             | <code>OperationId</code> , 16 big-endian bytes, raw                                         |
| <code>author</code>         | <code>AuthorId</code> , 16 big-endian bytes, raw                                            |
| <code>stamp</code>          | <code>OperationStamp</code> , 28 bytes (Section 6.5)                                        |
| <code>causal_context</code> | <code>CausalContext</code> (Section 6.5)                                                    |
| <code>transaction</code>    | option tag byte (0 absent / 1 present), then <code>TransactionId</code><br>16 BE if present |
| <code>payload</code>        | <code>OperationPayload</code> (Section 6.2)                                                 |

**Id-leads property.** The envelope’s first 16 bytes *are* the `OperationId`’s canonical bytes. This is load-bearing: the operation index (Chapter 7, Section 7.6) keys its entries on those leading 16 bytes without decoding envelopes, and the sanctioned partial read (*peek*) of an envelope is exactly its leading 16 bytes. Any future envelope revision **MUST** preserve id-leads or revise the operation index in the same schema-major step.

`EnvelopeHash` = `BLAKE3`(`MUSCENVH` || *envelope canonical bytes*) — the full 32-byte digest.

Envelopes are **encode-only** at this layer, deliberately: once committed, an envelope is stored and transported as opaque bytes and is never reconstructed into typed form by the storage layer; the peek above is the only sanctioned partial read. (Reducers decode payloads they authored or received through the ops layer’s own typed surface; the wire contract is that stored envelope bytes are preserved verbatim, since `EnvelopeHash` commits to them.)

## 6.2 OperationPayload and the Meta-Operations

`OperationPayload`: one discriminant byte, then the variant.

| Disc | Variant             | Payload                                                      |
|------|---------------------|--------------------------------------------------------------|
| 0    | Primitive           | an <code>OperationKind</code> (Section 6.3)                  |
| 1    | ResolveConflict     | target (ConflictId, 16 BE)    action (ResolutionAction)      |
| 2    | UndoTransaction     | target (TransactionId, 16 BE)    policy (UndoPolicy, 1 byte) |
| 3    | ResolveEquivocation | target (OperationId, 16 BE)    chosen (EnvelopeHash, 32)     |

Discriminants 0–2 are the ratified v1 values; 3 was appended by the catalog’s `RESOLVEEQUIVOCATION` (META-OPERATION) entry. The vocabulary is append-only: new meta-operations take  $\geq 4$ .

`UndoPolicy` is one byte: 0 `StrictInverse`, 1 `BestEffort`, 2 `Cascade`. `ResolutionAction` is one byte then an optional payload (app:bytes, core requirement req:semops:resolution-action-discriminants): 0 `AcceptLoser`, 1 `KeepWinner`, 2 `Override` (|| `OperationId` 16 BE), 3 `Reanchor` (|| `TypedObjectId`), 4 `Dismiss`, 5 `Registered` (|| `ResolutionRegistryId` 16 BE).

## 6.3 OperationKind: the Primitive Wire Discriminants

### REQUIREMENT 6.2

The `OperationKind` wire discriminant is one byte, golden-locked to the table below. The vocabulary is **append-only**: new primitive kinds take discriminants past 29; the assignments below never change.

Each row also pins the payload’s byte layout; the field *set and order* is the Operation Catalog’s (the cited section governs semantics). “`lp(T)`” means the `u32`-LE-length-prefixed `CanonicalValue` bytes of a `T` (Section 5.5); bare identifiers are raw 16 big-endian bytes; “`seq↑`” is a `push_seq` sequence sorted ascending by canonical bytes, and “`seq↑↑`” is the same sequence constrained to be *strictly* increasing — a set, not a multiset. A decoder **MUST** reject a `seq↑↑` containing a duplicate rather than normalize it away, since the duplicate is the only evidence that the writer and the reader disagree about the field’s type.

| Disc | Kind                     | Payload layout                                             | Catalog section          |
|------|--------------------------|------------------------------------------------------------|--------------------------|
| 0    | <code>InsertEvent</code> | <code>staff_instance</code> (16)    <code>lp(Event)</code> | <code>INSERTEVENT</code> |

| Disc | Kind                  | Payload layout                                                                                                                                       | Catalog section                |
|------|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| 1    | DeleteEvent           | event (16)    TupletCompensation                                                                                                                     | DELETEEVENT                    |
| 2    | RespellPitch          | pitch (16)    lp(PitchSpelling)                                                                                                                      | RESPELLPITCH                   |
| 3    | CreateCrossCutting    | CrossCuttingValue                                                                                                                                    | CREATECROSSCUTTING             |
| 4    | ChangeRegionTimeModel | region (16)    lp(RegionTimeModel)    seq <sup>↑</sup> (EventId)    PositionRemapping                                                                | CHANGERECTION<br>TIMEMODEL     |
| 5    | SetUserSystemBreak    | region (16)    lp(TimeAnchor)    present (bool, 1)                                                                                                   | SETUSERSYSTEMBREAK             |
| 6    | DeclareTransaction    | id (16)    label (NFC string)    option tag    TransactionCategory if present                                                                        | DECLARETRANSACTION             |
| 7    | Registered            | OperationKindRegistryId (16)    lp(opaque bytes)                                                                                                     | K1 — FRAMEWORK SLOTS           |
| 8    | ModifyEvent           | lp(Event)                                                                                                                                            | MODIFYEVENT                    |
| 9    | Transpose             | seq <sup>↑</sup> (PitchId)    chromatic_steps (i32 LE, 4). Frozen: the sequence is a multiset, not seq <sup>↑</sup>                                  | TRANPOSE                       |
| 10   | InsertIdentifiedPitch | event (16)    lp(IdentifiedPitch)                                                                                                                    | IDENTIFIED-PITCH<br>OPERATIONS |
| 11   | DeleteIdentifiedPitch | pitch (16)                                                                                                                                           | IDENTIFIED-PITCH<br>OPERATIONS |
| 12   | ModifyIdentifiedPitch | pitch (16)    lp(Pitch)                                                                                                                              | IDENTIFIED-PITCH<br>OPERATIONS |
| 13   | DeleteCrossCutting    | structure (TypedObjectId, 18/34)                                                                                                                     | DELETECROSSCUTTING             |
| 14   | ModifyCrossCutting    | CrossCuttingValue                                                                                                                                    | MODIFYCROSSCUTTING             |
| 15   | CreateRegion          | lp(Region)                                                                                                                                           | STRUCTURAL CONTAINERS          |
| 16   | DeleteRegion          | region (16)                                                                                                                                          | STRUCTURAL CONTAINERS          |
| 17   | CreateStaffInstance   | region (16)    lp(StaffInstance)                                                                                                                     | STRUCTURAL CONTAINERS          |
| 18   | DeleteStaffInstance   | staff_instance (16)                                                                                                                                  | STRUCTURAL CONTAINERS          |
| 19   | CreateVoice           | staff_instance (16)    lp(Voice)                                                                                                                     | STRUCTURAL CONTAINERS          |
| 20   | DeleteVoice           | voice (16)                                                                                                                                           | STRUCTURAL CONTAINERS          |
| 21   | SetMetadata           | lp(ScoreMetadata)                                                                                                                                    | SCORE SETTINGS                 |
| 22   | SetMetricGrid         | region (16)    option tag (0/1)    lp(MetricGrid) if present                                                                                         | SCORE SETTINGS                 |
| 23   | SetUserPageBreak      | region (16)    lp(TimeAnchor)    present (bool, 1)                                                                                                   | SCORE SETTINGS                 |
| 24   | CreateStaff           | lp(Staff)                                                                                                                                            | CREATESTAFF                    |
| 25   | SetTimeSignature      | region (16)    lp(TimeAnchor)    option tag (0/1)    lp(TimeSignature) if present                                                                    | METER AND TEMPO<br>OVERWRITES  |
| 26   | SetTempoSegment       | option tag    region (16) if present    lp(TimeAnchor)    option tag    lp(TempoSegment) if present                                                  | METER AND TEMPO<br>OVERWRITES  |
| 27   | SetStaffLayout        | staff_instance (16)    option tag    instrument_override (16) if present    option tag    lp(StaffLineConfiguration) if present    visible (bool, 1) | SETSTAFFLAYOUT                 |
| 28   | CreateRepeatStructure | lp(RepeatStructure) — the schema-major-2 layout, unconditionally (born at v2; Section 9.5)                                                           | REPEAT STRUCTURES              |
| 29   | DeleteRepeatStructure | repeat (16)                                                                                                                                          | REPEAT STRUCTURES              |
| 30   | TransposeInterval     | seq <sup>↑</sup> (PitchId)    diatonic_steps (i32 LE, 4)    chromatic_steps (i32 LE, 4)                                                              | TRANPOSEINTERVAL               |

Sub-vocabularies used above (one discriminant byte, then the listed payload):

**TupletCompensation.** 0 NotInTuplet (empty); 1 ReplaceWithRest || lp(Rest); 2 RewriteTuplets || seq<sup>↑</sup>(TupletId); 3 CascadeDeleteTuplets || seq<sup>↑</sup>(TupletId).

**CrossCuttingValue.** 0 Tie, 1 Slur, 2 Beam, 3 Spanner; in every case the tag is followed by `lp`(the typed structure’s `CanonicalValue` bytes).

**PositionRemapping.** 0 PreserveTime (empty); 1 Reassign `||` u32 LE entry count `||` entries, each `EventId` (16, raw) `||` `lp`(`MusicalPosition`), ascending by `EventId`.

**TransactionCategory.** 0 NoteEntry, 1 Structural, 2 Layout, 3 Import, 4 Registered `||` `OperationKindRegistryId` (16 BE) — governed by `app:bytes` (core requirement `req:semops:transaction-category`).

`OperationKind` is encode-only at the storage boundary, like the envelope that carries it.

## 6.4 OperationKindTag: a Separate Discriminant Space

### REQUIREMENT 6.3

`OperationKindTag` — the payload-free projection of an operation kind, stored by edit barriers (Chapter 8) — has its **own** discriminant space, assigned independently of Section 6.3, with both encode and validating decode. One byte; only `Registered` carries a payload (its `OperationKindRegistryId`, 16 big-endian bytes, total 17). Unknown discriminants and wrong lengths are decode errors. Append-only past 29.

| Disc | Tag                                | Disc | Tag                                |
|------|------------------------------------|------|------------------------------------|
| 0    | <code>InsertEvent</code>           | 12   | <code>DeleteStaffInstance</code>   |
| 1    | <code>DeleteEvent</code>           | 13   | <code>SetUserSystemBreak</code>    |
| 2    | <code>ModifyEvent</code>           | 14   | <code>SetUserPageBreak</code>      |
| 3    | <code>RespellPitch</code>          | 15   | <code>DeclareTransaction</code>    |
| 4    | <code>Transpose</code>             | 16   | <code>Registered</code>            |
| 5    | <code>CreateCrossCutting</code>    | 17   | <code>InsertIdentifiedPitch</code> |
| 6    | <code>DeleteCrossCutting</code>    | 18   | <code>DeleteIdentifiedPitch</code> |
| 7    | <code>ModifyCrossCutting</code>    | 19   | <code>ModifyIdentifiedPitch</code> |
| 8    | <code>ChangeRegionTimeModel</code> | 20   | <code>CreateVoice</code>           |
| 9    | <code>InsertRegion</code>          | 21   | <code>DeleteVoice</code>           |
| 10   | <code>DeleteRegion</code>          | 22   | <code>SetMetadata</code>           |
| 11   | <code>InsertStaffInstance</code>   | 23   | <code>SetMetricGrid</code>         |
| 24   | <code>InsertStaff</code>           | 25   | <code>SetTimeSignature</code>      |
| 26   | <code>SetTempoSegment</code>       | 27   | <code>SetStaffLayout</code>        |
| 28   | <code>CreateRepeatStructure</code> | 29   | <code>DeleteRepeatStructure</code> |
| 30   | <code>TransposeInterval</code>     |      |                                    |

**Naming mapping.** The kind-to-tag projection renames three pairs: `OperationKind::CreateRegion` projects to `OperationKindTag::InsertRegion`, and `OperationKind::CreateStaffInstance` projects to `OperationKindTag::InsertStaffInstance`, and `OperationKind::CreateStaff` projects to `OperationKindTag::InsertStaff`. The mismatch is intentional (the tag space predates the M2c naming) and is pinned here so barrier authors target the right tag. The repeat-structure pair projects name-verbatim (`CreateRepeatStructure/DeleteRepeatStructure`), as the cross-cutting tags do.

## 6.5 Stamps and Causal Contexts

HybridLogicalClock: `physical_time (WallClockTime, i64 LE, 8) || logical_counter (u32 LE, 4)`. OperationStamp: `hlc (12) || id (OperationId, 16 BE)` — **28 bytes** total, fixed width. The canonical reduction tuple orders stamps as *(physical\_time, logical\_counter, replica, counter)*, ascending; the replica comparison equals big-endian byte order.

CausalContext (dotted version vector):

| Field  | Encoding                                                                                        |
|--------|-------------------------------------------------------------------------------------------------|
| vector | u32 LE entry count    entries, each replica (8 BE)    counter (u64 LE, 8), ascending by replica |
| dots   | u32 LE count    OperationIds (16 BE each), ascending                                            |

The vector is *zero-based-floor* (P11-C7): an entry  $(r, c)$  covers counters  $0.. = c$  of replica  $r$ ; absence of a replica covers nothing.

## 6.6 Effects, Conflicts, Anomalies, and Materialized State

These types are canonical *materialized* facts: they appear in `MaterializedState`’s canonical bytes, so their discriminants and layouts are normative wire facts even though they are never stored inside envelopes. All decode validation is performed by the ops layer’s validating decoder; unknown discriminants, bad lengths, non-canonical orders, and trailing bytes are decode errors.

| Type                      | Layout (tag byte, then payload)                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OperationEffect           | 0 Applied (empty); 1 AppliedWithRepair    seq(RepairRecord); 2 Conflicted    ConflictId (16 BE); 3 TombstonedTarget    TypedObjectId; 4 NoOp    NoOpReason.                                                                                                                                                                                                                                         |
| NoOpReason                | 0 TargetTombstoned; 1 AlreadyApplied; 2 SupersededByLaterOperation    OperationId (16 BE); 3 PreconditionFailedUnderReduction    PreconditionFailureReason; 4 TransactionConflict.                                                                                                                                                                                                                  |
| PreconditionFailureReason | 0 TargetMissing; 1 TargetTombstoned; 2 WrongRegionTimeModel; 3 TupleletCompensationInvalid; 4 EventDurationInvalid; 5 PositionOutsideRegion; 6 PitchSpaceMismatch; 7 VoiceMissing; 8 ExtensionPrecondition    id (16 BE); 9 Registered    id (16 BE); 10 ContainerNotEmpty; 11 TempoMapMalformed; 12 SystemDerivedContentImmutable (Pass 12, P12-K3); 13 RecreateContentMismatch (Pass 12, P12-K9). |
| RepairRecord              | (struct) kind (RepairKind)    target (TypedObjectId).                                                                                                                                                                                                                                                                                                                                               |
| RepairKind                | 0 Reanchored    from (TypedObjectId)    to (TypedObjectId)    ReanchorReason; 1 SpannerTruncated    seq(TypedObjectId); 2 Orphaned; 3 CascadeDeleted; 4 AttachmentTombstoned; 5 VoicePromoted    from (VoiceId, 16 BE)    to (VoiceId, 16 BE); 6 TupleletCompensated    TupleletCompensationKind; 7 Registered    id (16 BE).                                                                       |
| ReanchorReason            | 0 SameVoiceNearer; 1 SameStaffInstanceNearer; 2 SameStaffNearer; 3 SameRegionNearer; 4 ExplicitFallback; 5 DeclaredByExtension    id (16 BE); 6 SameCanvasNearer (Pass 12, P12-C4: semantically the rank-4 proximity variant; appended at 6 because 5 was already owned by DeclaredByExtension).                                                                                                    |
| TupleletCompensationKind  | 0 ReplaceWithRest; 1 RewriteTuplelets; 2 CascadeDeleteTuplelets. (No payload; distinct from the payload-carrying <code>TupleletCompensation</code> of Section 6.3.)                                                                                                                                                                                                                                 |

| Type                    | Layout (tag byte, then payload)                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PendingReason           | tag    blocker (OperationId, 16 BE) in every variant: 0 MissingCausalPredecessor; 1 DependsOnEquivocated; 2 DependsOnExcluded; 3 DependsOnPending; 4 HaltedBySystemCollision.                                                                                                                                                                                                                                                                                                       |
| ConflictKind            | 0 StructuralFieldCollision    winner (16 BE)    loser (16 BE)    FieldPath (NFC string); 1 TransactionConflict    transaction (16 BE)    seq <sup>↑</sup> (OperationId); 2 TombstonedTarget    TypedObjectId    OperationId (16 BE); 3 ReanchorFailure    original_referent (TypedObjectId)    referencing_object (TypedObjectId); 4 TimeModelMigrationFailure    region (16 BE)    seq <sup>↑</sup> (TypedObjectId); 5 ExtensionConflict    kind_id (16 BE)    lp(opaque details). |
| ConflictResolutionState | 0 Unresolved; 1 Resolved    by (16 BE)    ResolutionAction; 2 Dismissed    by (16 BE).                                                                                                                                                                                                                                                                                                                                                                                              |
| ConflictRecord          | (struct) id (ConflictId, 16 BE)    seq(caused_by: OperationId, stored sorted)    kind    seq(affected_objects: TypedObjectId, stored sorted)    resolution_state.                                                                                                                                                                                                                                                                                                                   |
| IntegrityAnomaly        | (struct) id (IntegrityAnomalyId, 16 BE)    kind.                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| IntegrityAnomalyKind    | 0 SystemIdentifierCollision    ObjectKind    colliding_counter (u64 LE)    the two input-set blobs, each lp(bytes), emitted lo ≤ hi by byte comparison; 1 OperationSlotEquivocated    OperationId (16 BE); 2 ReplicaStreamQuarantined    replica (8 BE)    first_bad_counter (u64 LE); 3 Registered    id (16 BE).                                                                                                                                                                  |
| AnomalousReplicaSegment | (struct) replica (8 BE)    first_bad_counter (u64 LE)    ReplicaAnomalyReason (0 HlcMonotonicityViolation    two OperationIds; 1 Registered    id 16 BE)    seq(excluded: OperationId, ascending counter).                                                                                                                                                                                                                                                                          |
| FieldPath               | an NFC string (u32 LE length    UTF-8).                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| ObjectState             | 0 Live (empty); 1 Tombstoned    deleted_by (OperationId, 16 BE)    minted_by (OperationId, 16 BE).                                                                                                                                                                                                                                                                                                                                                                                  |
| ObjectKind (ops)        | 0 Voice; 1 Pitch; 2 Registered    OperationKindRegistryId (16 BE) — governed by app:bytes (core requirement req:graph:object-kind-vocab). Distinct from the barrier layer’s 2-byte ObjectKind (Chapter 8).                                                                                                                                                                                                                                                                          |

### 6.6.1 MaterializedState

The canonical bytes of a reduction’s materialized state are eight sections, concatenated; every section is u32-LE-counted (the conflict registry carries its own count) and in the stated normative order, so the bytes are identical across any permutation of the input operation set:

1. **effects** — count, then per entry OperationId (16 BE, raw) || lp(OperationEffect); in canonical reduction order;
2. **conflict registry** — a seq of lp(ConflictRecord), ascending ConflictId;
3. **anomalies** — count, then each lp(IntegrityAnomaly), ascending IntegrityAnomalyId;
4. **objects** — count, then per entry TypedObjectId (raw) || ObjectState, ascending TypedObjectId;
5. **spellings** — count, then per entry PitchId (16 BE, raw) || lp(PitchSpelling), ascending PitchId;
6. **breaks** — count, then per entry RegionId (16 BE, raw) || MusicalPosition (raw RationalTime form, self-describing, *not* leaf-framed) || present (bool, 1); ascending (region, position);
7. **page breaks** — same shape and order as breaks;

8. **pending** — count, then per entry `OperationId` (16 BE, raw) || `PendingReason`, ascending `OperationId`.

Decode is fully validating (orders, tags, lengths, trailing bytes) and rejects non-canonical input.

## 6.7 vo Payloads Have No Wire Form

The prototype's vo identifier-projection payload types are an **in-memory migration guard only**: they carry no canonical encoding, and no vo wire form was ever shipped. The `vo→v1` migration is *semantic* — typed value reconstruction, specified in the Operation Catalog's `vo → v1 PAYLOAD MIGRATION` chapter — not a byte-level translation, and nothing in this document defines bytes for a vo payload.

# Bundle Physical Layout

The physical file: the fixed prelude, chunks, blocks, the manifest, and the operation index. This chapter **ratifies** the bundle crate’s provisional encodings (its `DECISIONS` entries `P11-D2`, `P11-D4`, and `P11-D5` pending-companion status is discharged) and the operation-index byte form (`P12-D1`). Structural *semantics* — superblock selection, commit protocol, recovery, retention — are core specification Chapter 8; this chapter pins bytes only. Regime (a) applies except where a table says otherwise.

## 7.1 The Fixed Header (64 bytes)

Offset 0 of every bundle. Written at creation, never rewritten within a format major version. All integers little-endian.

| Range  | Field                     | Value / rule                                   |
|--------|---------------------------|------------------------------------------------|
| 0..8   | magic                     | MUSCBND\0 (8 bytes, ASCII, trailing NUL)       |
| 8..10  | format_major (u16)        | 0 in this version; readers reject other values |
| 10..12 | format_minor (u16)        | 1 written in this version                      |
| 12..16 | header_length (u32)       | 64; readers reject other values                |
| 16..24 | superblock_a_offset (u64) | 64; readers reject other values                |
| 24..32 | superblock_b_offset (u64) | 320; readers reject other values               |
| 32..48 | file_uuid                 | 16 opaque bytes                                |
| 48..60 | reserved                  | written zero; ignored on read                  |
| 60..64 | header_crc                | CRC-32C of bytes 0..60, u32 LE                 |

Readers **MUST** verify the magic and the CRC before consulting any other part of the file.

## 7.2 The Superblock Slots (256 bytes each)

Two slots, at offsets 64 (A) and 320 (B); the only mutable on-disk objects. Selection *semantics* (highest valid committed generation, the equal-generation rule, torn-write fallback) are core Chapter 8, `SUPERBLOCK SELECTION`; the bytes:

| Range | Field      | Encoding                 |
|-------|------------|--------------------------|
| 0..8  | magic      | MUSCSUPR (8 ASCII bytes) |
| 8..16 | generation | u64 LE                   |

| Range    | Field                       | Encoding                                                                                                                                                                                                                                     |
|----------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 16..24   | manifest_offset             | u64 LE                                                                                                                                                                                                                                       |
| 24..32   | manifest_length             | u64 LE (also the uncompressed length; the manifest is mandatorily uncompressed)                                                                                                                                                              |
| 32..64   | manifest_hash               | 32 raw digest bytes                                                                                                                                                                                                                          |
| 64..68   | manifest_schema_version     | SchemaVersion (u16 LE    u16 LE)                                                                                                                                                                                                             |
| 68..72   | reduction_algorithm_version | u32 LE                                                                                                                                                                                                                                       |
| 72..92   | profile_id                  | 20 bytes: u32 LE discriminant (0 Full, 1 ReadOnly, 2 Lite, 3 Custom)    ProfileRegistryId (16 raw bytes; writers <b>MUST</b> emit zero unless Custom, readers ignore it unless Custom). Core requirement req:format:profileid-discriminants. |
| 92..100  | commit_state                | two u32 LE words: Committed = (0,0); any tag $\neq 0$ decodes as Reserved(value) and the slot is invalid for ordinary selection. Writers in this version <b>MUST</b> produce only Committed.                                                 |
| 100..108 | commit_timestamp            | i64 LE nanoseconds; advisory — selection never consults it                                                                                                                                                                                   |
| 108..252 | reserved                    | written zero; ignored on read                                                                                                                                                                                                                |
| 252..256 | superblock_crc              | CRC-32C of bytes 0..252, u32 LE                                                                                                                                                                                                              |

### 7.3 Chunks

The bundle body begins at `BODY_START = 320 + 256 = 576`. A chunk’s on-disk form is its raw (possibly compressed) payload bytes at a byte offset — **no per-chunk length prefix, magic, or trailer on disk**; the offset, lengths, compression, and hash live in the referencing `ChunkRef`. A chunk reference whose offset is below `BODY_START` is malformed.

**Content hash preimage** (core Chapter 8, DOMAIN-SEPARATED PREIMAGES; governs identity):

$$\text{BLAKE3}(\text{domain} \parallel \text{kind}(1) \parallel \text{schema}(4) \parallel \text{uncompressed\_length}(\text{u64 LE}) \parallel \text{payload})$$

where *domain* is `MUSCMANI` for Manifest chunks and `MUSCCHNK` otherwise. Compression is *not* in the preimage. Blob chunks are addressed by the bare `BlobId` form instead (Chapter 4).

**ChunkKind** is a single byte, a **closed** vocabulary (no Registered variant; core requirement req:format:chunkkind-discriminants): 0 OperationEnvelopeBlock, 1 OperationIndex, 2 Snapshot, 3 Blob, 4 ExtensionData, 5 TextProjection, 6 LayoutCache, 7 IntegrityIndex, 8 Manifest.

**CompressionAlgorithm** is a fixed two bytes, discriminant || parameter: None = (0,0); Zstd{level} = (1,level); Reserved(*v*) = (2,*v*). None is *not* a bare tag — its zero parameter byte is always present.

**REQUIREMENT 7.1**

A decoder **MUST** reject a `CompressionAlgorithm` whose discriminant is 0 (`None`) and whose parameter byte is non-zero. It **MUST NOT** ignore the byte, and **MUST NOT** normalize it to zero.

**RATIONALE**

Ignoring it made the codec *lenient*, and therefore non-injective: (0, 255) and (0, 0) decoded to one value, which re-encodes to the latter. Every structure embedding a `ChunkRef` inherited that. A whole-value re-encode-and-compare guard hides it — the manifest has one — but the operation index does not, and so accepted two distinct payloads for one index while its own contract promised to reject non-canonical bytes rather than normalize them. Content-addressing rests on distinct bytes meaning distinct values.

No conforming writer emits a non-zero parameter for `None`, so this rejects only corrupt or adversarial input; no existing file changes meaning. Found by the wire-decode fuzzer (Push 5, P3); supersedes the core specification’s earlier “present but zero, and ignored on read”.

**Read rules.** Writers in this format version emit `None` only; *reading* `zstd` at any level is a conformance **MUST**. A `zstd` payload is decompressed into a buffer sized *exactly* by the declared `uncompressed_length` (checked against the reader’s resource policy before allocation): a stream that would exceed the declaration, ends short of it, or carries trailing garbage is corruption. After decompression the content hash is verified over the uncompressed bytes, and `id = hash` is checked. A chunk whose declared schema major differs from the reader’s supported major is rejected (Chapter 9). *Reserved* compression is rejected. The **manifest chunk is mandatorily uncompressed**: a compressed manifest reference is rejected before any bytes are read. The reference reader bounds any single chunk at 256 MiB (`MAX_CHUNK_BYTES`); the bound is reader policy, not a wire fact.

**7.3.1 ChunkRef (95 bytes)**

| Range  | Field                            | Encoding                                        |
|--------|----------------------------------|-------------------------------------------------|
| 0..32  | <code>id</code>                  | 32 raw digest bytes                             |
| 32..33 | <code>kind</code>                | <code>ChunkKind</code> byte                     |
| 33..37 | <code>schema_version</code>      | u16 LE    u16 LE                                |
| 37..45 | <code>offset</code>              | u64 LE                                          |
| 45..53 | <code>compressed_length</code>   | u64 LE                                          |
| 53..61 | <code>uncompressed_length</code> | u64 LE                                          |
| 61..63 | <code>compression</code>         | 2 bytes                                         |
| 63..95 | <code>hash</code>                | 32 raw digest bytes (restates <code>id</code> ) |

The canonical sort key for chunk-reference collections is (*kind discriminant*, *hash*, *offset*), ascending — the Appendix-D chunk-reference order. Every set-valued `ChunkRef` field in this chapter is sorted by it and deduplicated before encoding.

## 7.4 Operation-Envelope Block Payloads

The uncompressed payload of an `OperationEnvelopeBlock` chunk:

`u32 LE envelope count || per envelope {u32 LE byte length || envelope bytes}`

Envelope bytes are opaque to the bundle (their internal form is Chapter 6). Each envelope’s offset — the byte offset of its *first content byte* within the uncompressed block payload, with its length prefix at *offset* − 4 — is deterministically recoverable from this framing alone; it is the coordinate the operation index records. Writers **SHOULD** begin a new block rather than let an uncompressed payload exceed 1 MiB (an individual envelope larger than that occupies its own block); readers **MUST** accept blocks up to the active profile’s bound (default 64 MiB). Block boundaries are storage artifacts: the envelope set is the union across blocks.

## 7.5 The Manifest

On-disk manifest chunk payload:

`manifest_id (16 bytes, u128 little-endian) || body`

(The little-endian on-disk field is a deliberate, golden-locked quirk: the *derivation* truncates big-endian like every content-derived id, but the bundle codec serializes the resulting `u128` under its little-endian integer rule.)

### REQUIREMENT 7.2

Restating core requirement `req:format:manifest-id`, which governs:

`ManifestId = trunc128(BLAKE3(MUSCMNIF || document_id || generation || body))`

with *document\_id* as its 16 raw bytes, *generation* as `u64` little-endian, and *body* the complete manifest body below (which itself opens with *document\_id* and *generation* — the double commitment is intentional and locked). Writers re-derive the id from the body at encode time; decoders verify it.

The **body** is a positional struct of thirteen fields:

| # | Field                                  | Encoding and canonical order                                                                                                               |
|---|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <code>document_id</code>               | 16 raw bytes                                                                                                                               |
| 2 | <code>lineage_id</code>                | option tag    16 raw bytes if present                                                                                                      |
| 3 | <code>generation</code>                | <code>u64 LE</code>                                                                                                                        |
| 4 | <code>operation_roots</code>           | <code>u32 LE count</code>    <code>ChunkRefs</code> , sorted by the canonical chunk-reference key, deduplicated                            |
| 5 | <code>operation_block_summaries</code> | <code>u32 LE count</code>    entries { <code>chunk_id</code> (32)    <code>OperationBlockSummary</code> }, ascending <code>chunk_id</code> |
| 6 | <code>operation_index_root</code>      | option tag    <code>ChunkRef</code>                                                                                                        |
| 7 | <code>canonical_base</code>            | option tag    <code>SnapshotRef</code>                                                                                                     |
| 8 | <code>acceleration_snapshots</code>    | <code>u32 LE count</code>    <code>SnapshotRefs</code> , sorted ascending by their full encoded bytes, deduplicated                        |
| 9 | <code>blob_roots</code>                | <code>u32 LE count</code>    <code>BlobRefs</code> , sorted ascending by encoded bytes, deduplicated                                       |

| #  | Field                  | Encoding and canonical order                                                                                                       |
|----|------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 10 | profile_declarations   | u32 LE count    ProfileDeclarations, sorted by the key (profile_id,version) with SemVer compared <i>numerically</i> , deduplicated |
| 11 | extension_declarations | u32 LE count    ExtensionDeclarations, sorted by (extension_id,version), SemVer <b>numeric</b> , deduplicated                      |
| 12 | text_projection_root   | option tag    ChunkRef                                                                                                             |
| 13 | integrity_root         | option tag    ChunkRef                                                                                                             |

Sub-records, positionally:

**OperationBlockSummary** = dvv\_summary (FrontierBytes: u32-LE-prefixed opaque bytes) || min\_stamp (u32-LE-prefixed opaque stamp bytes) || max\_stamp (same). The contents are ops-computed (Chapter 6); the bundle carries them verbatim.

**SnapshotRef** = snapshot\_id (16 raw) || covers\_causal\_frontier (FrontierBytes) || reduction\_algorithm\_version (u32 LE) || profile\_id (20 bytes, as in the superblock) || root (ChunkRef, 95) || hash (32 raw).

**BlobRef** = blob\_id (32 raw) || media\_type (u32-LE-prefixed ASCII; decode validates a well-formed RFC 6838 type/subtype **restricted name**) || offset (u64 LE) || compressed\_length (u64 LE) || uncompressed\_length (u64 LE) || compression (2) || hash (32 raw) || option tag || declared\_max\_uncompressed\_length (u64 LE) if present.

**ProfileDeclaration** = profile\_id (20) || version (SemVer, 12) || constraints (ProfileConstraints = max\_uncompressed\_block\_size (u64 LE) || RetentionPolicy). RetentionPolicy = retain\_previous\_manifests (u32 LE) || option tag || retain\_duration (i64 LE) if present || retain\_named\_checkpoints (bool, 1).

**ExtensionDeclaration** = extension\_id (16 raw) || version (SemVer, 12) || required (bool, 1) || u32 LE count || preserved\_chunk\_roots (ChunkRefs, canonical order, deduplicated) || affected\_object\_kinds (u32-LE-prefixed opaque blob) || edit\_barriers (u32-LE-prefixed opaque blob). The two blobs' internal form is Chapter 8; the bundle preserves them verbatim.

### REQUIREMENT 7.3

**Reject non-canonical manifests.** A decoder **MUST** accept a manifest payload only if re-encoding the decoded manifest reproduces the input byte-for-byte. This single check subsumes: the stored `manifest_id` matching the body-derived id, every set-valued vector being in its canonical order with no duplicates, and every optional field using presence byte 0/1. Unsorted, duplicated, or wrongly-identified manifest bytes are malformed — never re-sorted or repaired.

## 7.6 The Operation Index

### REQUIREMENT 7.4

**Operation-index payload (ratifies P12-D1).** The uncompressed payload of an `OperationIndex` chunk is:

u32 LE `block_count` || that many `ChunkRefs` (95 bytes each, strictly ascending canonical order) || u32 LE `entry_count` || that many entries, each {id (16 raw operation-id bytes) || block (u32 LE ordinal into the block vector) || offset (u32 LE)}, strictly ascending by id bytes.

The empty index is exactly 8 zero bytes. `offset` is the byte offset of the envelope’s *first content byte* within the uncompressed payload of the named block (the envelope’s u32 length prefix sits at `offset - 4`) — well-defined by the id-leads property (Requirement 6.1) and the block framing (Section 7.4).

**One slot per id.** Strict ascent makes duplicate ids unrepresentable: an index payload containing two entries with equal id bytes is malformed. Equivocation candidates (multiple envelopes claiming one `OperationId`) live *inside* the operation set’s slot model (core Chapter 6), never as duplicate index rows.

**Staleness.** An index *covers* a manifest iff its block vector equals the manifest’s deduplicated, canonically sorted `operation_roots` under **full `ChunkRef` equality** — every field, not just the content hash, since the index hands out its stored references for reading. A reader that finds the index stale **MUST** reject it and rebuild from the blocks.

A rejected or stale index is **never bundle corruption**: the index is a non-canonical accelerator, and rebuild-from-blocks is always sound.

Decoders **MUST** reject: unsorted or duplicated blocks or entry ids, a block reference whose kind is not `OperationEnvelopeBlock`, an entry whose block ordinal is out of range, and trailing bytes.

The core specification’s commit-time **SHOULD** — refresh the index when the operation log has “grown significantly” since it was built — has an **implementation-defined** threshold in vo: this document deliberately does not pin a number, and conforming writers may use any policy (including always or never refreshing at commit), because the index is non-canonical.

### OPEN QUESTION

Whether to pin a normative refresh threshold (e.g. a block-count or byte-size ratio) or leave it permanently implementation-defined. Leaving it open costs only lookup performance on stale-index bundles; pinning it too early would constrain writers for no interoperability gain. Revisit when a second writer implementation exists.

# Extension Declaration Blobs and Edit Barriers

The two opaque blobs a manifest `ExtensionDeclaration` carries — `affected_object_kinds` and `edit_barriers` — have their internal byte form pinned here. This chapter **ratifies** P12-E1 (the blob and barrier-tree byte form), P12-E2 (the recursion bound), and P12-E3 (the barrier `ObjectKind` representation). Barrier *semantics* — scope matching, prohibited-edit evaluation, conservative treatment of unknown extensions — are core Chapter 8 (FORWARD COMPATIBILITY AND EDIT BARRIERS); this chapter pins bytes only.

These layouts are **regime (b)**: all counts and length prefixes are `u64` little-endian — a deliberate, golden-locked divergence from the `u32` regime (Section 2.1). They **keep this form in schema major 1**: unlike the resolved-layout surface, these blobs are carried *opaquely inside the canonical manifest* (Section 7.5), which stays major 0, so a major-0 reader must continue to parse them as `u64`. Their unification waits for a future major that revises the manifest (Section 9.4).

## OPEN QUESTION

**Unsafe-edit tombstone encoding (from P12-E5).** Core Chapter 8 now ratifies the unsafe-edit tombstone *semantics* (immediate deactivation of the crossed extension’s remaining barriers; a durable record **MUST** land at the next commit; `required = true` interaction). The manifest-side *byte encoding* of that record is unresolved — and constrained: the manifest schema is frozen at major 0, so the record cannot be a new manifest field. Candidate channels: a marker inside the extension-declaration blob layer (this chapter’s regime-(b) surface), or a dedicated chunk kind referenced from `preserved_chunk_roots`. To be designed in the next bundle-format tranche; the editor-side producer (`extensions_requiring_tombstone()`) already exists.

## 8.1 Framing

**element** `elem(x)` = `u64` LE byte length || *x*’s canonical bytes.

**set** `set(xs)` = elements’ canonical byte strings, sorted ascending byte-lexicographic and *deduplicated*; then `u64` LE element count || each element as `elem`. Order and repetition in the source collection cannot affect the bytes; decoders reject unsorted or duplicated elements.

**list** `list(xs) = u64 LE count || each element as elem, order-significant (used only for condition subtrees).`

#### REQUIREMENT 8.1

**The two blobs (ratifies P12-E1).** The `affected_object_kinds` blob is set of barrier `ObjectKind` elements; the `edit_barriers` blob is set of `EditBarrier` elements. Both decode under reject-never-normalize with trailing bytes forbidden.

#### REQUIREMENT 8.2

**Barrier `ObjectKind` (ratifies P12-E3).** A barrier `ObjectKind` is the `TypedObjectId` discriminant of Section 3.2, encoded as **2 little-endian bytes**. Decode is **open-value**: every `u16` decodes successfully — an unknown kind (a future core kind or an extension-registered kind) is a *value*, not a decode branch, and simply never matches any object the reader knows. This is the format’s forward-compatibility stance for barrier kinds: unknown kinds are preserved and re-emitted verbatim, never dropped or rejected.

## 8.2 EditBarrier

An `EditBarrier`’s canonical bytes, positionally:

```
scope || set(affected_object_kinds: ObjectKind) || set(prohibited_operation_kinds:
               OperationKindTag, Section 6.4) || condition
```

`BarrierScope`: one discriminant byte, then the payload:

| Disc | Variant       | Payload                                                                                                |
|------|---------------|--------------------------------------------------------------------------------------------------------|
| 0    | WholeScore    | (none)                                                                                                 |
| 1    | Region        | RegionId, 16 big-endian bytes                                                                          |
| 2    | StaffInstance | StaffInstanceId, 16 BE                                                                                 |
| 3    | AnalysisLayer | AnalysisLayerId, 16 BE                                                                                 |
| 4    | ObjectSet     | set of <code>TypedObjectId</code> canonical bytes (18/34 each)                                         |
| 5    | PitchSpace    | <code>u64 LE byte length    the pitch-space id’s NFC UTF-8 bytes; decoders reject non-NFC input</code> |
| 6    | TuningContext | (none)                                                                                                 |
| 7    | Registered    | <code>BarrierScopeRegistryId, u128 little-endian (16 bytes)</code>                                     |

`BarrierCondition`: one discriminant byte, then the payload; the tree is recursive:

| Disc | Variant                | Payload                                                                      |
|------|------------------------|------------------------------------------------------------------------------|
| 0    | Always                 | (none)                                                                       |
| 1    | ObjectExists           | <code>TypedObjectId</code>                                                   |
| 2    | ObjectHasExtensionData | <code>object (TypedObjectId)    extension (ExtensionRef, u128 LE, 16)</code> |
| 3    | All                    | list of conditions                                                           |
| 4    | Any                    | list of conditions                                                           |

| Disc | Variant    | Payload                                  |
|------|------------|------------------------------------------|
| 5    | Not        | one elem-framed condition                |
| 6    | Registered | BarrierConditionRegistryId, u128 LE (16) |

Note the endianness: the barrier layer’s registry identifiers (`BarrierScopeRegistryId`, `BarrierConditionRegistryId`, `ExtensionRef`) encode u128 *little-endian*, unlike the big-endian identifier family of Chapter 3. This is part of the golden-locked regime-(b) surface.

#### REQUIREMENT 8.3

**Recursion bound (ratifies P12-E2).** `MAX_CONDITION_DEPTH = 64` is the normative bound on `BarrierCondition` nesting. Decoders **MUST** reject a condition tree nested deeper than 64 levels of `All/Any/Not`; writers **MUST NOT** emit one. The bound exists so adversarial bytes cannot drive unbounded recursion; real barriers are depth 1–2.

# Schema Evolution

---

The core specification delegates schema evolution to this document (core Chapter 8, `SCHEMA VERSIONING`: “*Schema evolution is governed by the Binary Format companion specification, which defines the wire encoding for each schema version*”). This chapter defines the wire rules for schema evolution. Chapters 5–8 specify the schema-major-0 layouts; Section 9.4 specifies the schema-major-1 delta and the migration between them, and Section 9.5 the schema-major-2 delta and its migration from major 1.

## 9.1 The Chunk-Level Gate

Every chunk declares a `SchemaVersion` (major, minor) in its `ChunkRef` and the superblock declares the manifest’s. The gate:

- **Major** = incompatible. A reader **MUST** reject a chunk whose schema major it does not support. A reader supports a contiguous accept-set  $[\text{MIN}, \text{MAX}]$  of majors and rejects any chunk whose major falls *outside*  $[\text{MIN}, \text{MAX}]$  — too new (above `MAX`) or, once `MIN` rises past a retired major, too old (Section 9.4). Three majors are defined: 0, 1 (Section 9.4), and 2 (Section 9.5); the reference implementation’s accept-set is  $\{0, 1, 2\}$  (`MIN` = 0, `MAX` = 2).
- **Minor** = additive. `vo` readers verify the major only; the minor is a *record*, not a gate — but it is a mandatory record: a writer **MUST** raise the chunk schema minor when it emits any discriminant appended after the minor it otherwise declares, so that a decode failure on an unknown appended discriminant is attributable to a version skew rather than corruption.

## 9.2 What “Additive” Means Here

Under the frozen-layout rule (Requirement 5.3), positional struct layouts admit *no* additive change: there is no optional-field or trailing-field mechanism at the value level. In particular, and stated honestly: **the manifest body cannot grow a fourteenth field within major 0**. The manifest is a positional struct whose decoder rejects trailing bytes, so manifest-body extension is major-gated in `vo` exactly like every other struct.

Adding a field is a **MAJOR** change *regardless of its type*: an `Option` field is not “optional” in the wire sense — it still occupies a new positional slot (a presence byte, then the payload when present), which shifts every subsequent field and rejects under the trailing-bytes rule. There

is no “downgrade to minor” for an `Option` addition. The first exercise of this is schema major 1 (Section 9.4), which adds `Instrument.range` (an `Option`) among other fields.

The *only* minor-additive mechanism in schema major 0 is **appending discriminants to open vocabularies**:

- `OperationKind`: append at  $\geq 30$  (Requirement 6.2; the Phase-3 tranche took 24–27 under this mechanism, and the repeat pair 28/29 in the schema-major-2 revision — the discriminant space is one append-only table across majors, but a kind’s *block stamp* follows minimal stamping over its payload: 28’s payload embeds a v2 layout, so its blocks stamp major 2, Section 9.5);
- `OperationKindTag`: append at  $\geq 30$  (Requirement 6.3);
- `OperationPayload`: append at  $\geq 4$  (Section 6.2);
- the value-layer unions enumerated as closed in Requirement 5.3 may gain appended variants only through a ratified revision of this document, also minor-additive.

`ChunkKind` is **closed** — it has no `Registered` variant and no append story inside major 0, because its discriminant enters every chunk’s hash preimage; a new chunk kind is a format-major event.

Separately, many vocabularies carry a `Registered escape variant` (`RepairKind`, `ReanchorReason`, `PreconditionFailureReason`, `IntegrityAnomalyKind`, `ReplicaAnomalyReason`, `TransactionCategory`, `ResolutionAction`, `ConflictKind` via `ExtensionConflict`, `BarrierScope/BarrierCondition`, `TieClass`, `StaffGroupKind`, `PitchSpacePosition`, `SpellingNominal`, `TypedObjectId`, and the barrier `ObjectKind`’s open value space). Extension through an escape variant is **not** a schema change at all: the wire form is already defined.

### 9.3 Migration Discipline

A schema-MAJOR bump **MUST** ship with a migration documented in this document’s revision history (Chapter 12): what changed, byte-for-byte, and the deterministic translation from the old form. The precedent is the `v0`→`v1` operation-payload migration, which lives in the Operation Catalog (`v0` → `v1` PAYLOAD MIGRATION) because it was semantic rather than byte-level (Section 6.7); byte-level migrations belong here.

The `u64/u32` length-prefix divergence between regimes (a) and (b) (Section 2.1), left open in earlier revisions for a future major, is **partly resolved in schema major 1** (Section 9.4): the first major bump, scheduled for data-model reasons, carries the unification for the independent non-canonical resolved-layout surface. The barrier/extension blobs remain regime (b) because they ride the canonical manifest, which this bump keeps at major 0; their unification stays open for a manifest-revising major.

### 9.4 Schema Major 1

Schema major 1 is the first data-model expansion major, defined here in full: the changed value layouts, the length-prefix unification, and the byte-for-byte migration from major 0. It

bundles every change that was deferred to “the next major” so that one coordinated migration discharges them together.

#### 9.4.1 Where the changed fields reach

The three added fields land in different chunk classes, and the major is assigned **per payload type**, not per chunk kind:

- `Canvas.layout_defaults` and `Instrument.range` appear *only* in the acceleration-cache full-Score snapshot (a `ChunkKind::Snapshot`). No operation payload embeds a `Canvas` or an `Instrument` value (there is no `CreateCanvas/CreateInstrument`; `canvas` is the inline genesis singleton and instruments live in score genesis), so these two are confined to that one **non-canonical** chunk.
- `Region.permits_spanning_slurs` reaches the snapshot *and* the **canonical operation layer**: `CreateRegion` embeds the full `Region` value (Section 6.2), so its v1 payload carries the v1 `Region` layout. An operation-envelope block containing a `CreateRegion` therefore encodes v1 bytes and is stamped **major 1**; a block with no such operation stays **major 0**. Operation-envelope blocks are canonical, so this is a canonical operation-layer change, not merely a cache change.
- `ChunkKind::Snapshot` is itself payload-polymorphic: the same kind carries *either* an acceleration-cache full `Score` (as above, stamped major 1) *or* the canonical-base `MaterializedState`, disambiguated by manifest role. The canonical base embeds no `Canvas`, `Instrument`, or `Region` value, so it is unchanged and stays **major 0**. Its content-hash preimage (Section 7.3) includes the schema version, so re-stamping the unchanged base at major 1 would change its chunk id and churn every `SnapshotRef` for zero layout benefit. A writer **MUST NOT** do so; a conformance test **SHOULD** assert the canonical base is byte-identical across the bump.

#### 9.4.2 Cross-major reader behaviour

Two rules follow from the chunk classes above and the core specification’s `SCHEMA VERSIONING` canonical/non-canonical distinction:

- **Non-canonical chunks — discard and regenerate.** The layout caches and the acceleration snapshot are non-canonical. A reader meeting a foreign-major one **MAY** discard and regenerate it rather than decode it: a major-1 reader regenerates a major-0 layout cache instead of carrying a frozen old-width decoder, and it migrates a major-0 acceleration `Score` snapshot on read (cheaper than replaying the operation log).
- **Canonical chunks — parse or open read-only.** The manifest and the canonical base stay major 0 and are always parseable. A major-1 op block is canonical: a major-0-only reader **MUST NOT** discard it (that would fork canonical state) — it opens the bundle in read-only preservation mode, having read the major-0 base and manifest but being unable to replay the v1 `CreateRegion` operations. A major-1 reader migrates such a block on read (below).

So a major-0-only reader opens a major-1 bundle *fully* only when the bundle contains no v1 `CreateRegion` operation; otherwise it opens read-only. It always reads the canonical base and manifest, and always discards the higher-major non-canonical caches.

### 9.4.3 The accept-set gate

A reader supports a contiguous set of majors  $[\text{MIN}, \text{MAX}]$  (at the major-1 revision the reference implementation's set was  $\{0, 1\}$ ; the *current* set is stated once, in Section 9.1, and is not repeated here). The chunk-level gate rejects a chunk whose major falls outside  $[\text{MIN}, \text{MAX}]$  — too new above  $\text{MAX}$ , or too old below  $\text{MIN}$  once a reader drops support for a retired major; it no longer rejects on inequality with a single supported major.

### 9.4.4 Changed and new value layouts

These extend Chapter 5's regime (a) rules. As there, **the wire form is the reference implementation's struct layout** — which is a reduced subset of the core specification's fuller data model (the code's `Instrument` carries a subset of the model's fields, etc.). The wire form ratifies the code, not the model; a struct's v1 layout **appends** its new field(s) after its existing major-0 fields:

- `Canvas = regions || layout_defaults`, where `CanvasLayoutDefaults = page_size (CanvasSize = width || height, two CanonicalF64 leaves) || margins (CanvasMargins = four CanonicalF64 leaves: top, right, bottom, left)`.
- `Instrument = id || name || range (Option<PitchRange>: a presence byte, then for Some a PitchRange = lowest || highest, each a Pitch under this chapter's rules)`.
- `Region` appends `permits_spanning_slurs`, one bare bool byte (0/1), after its major-0 fields.
- **Operation layer:** the `CreateRegion` payload embeds a length-prefixed `Region` (Section 6.2), so its v1 form carries the v1 `Region` layout (the appended bool inside the embedded value). This is the one *canonical* byte change in major 1; the operation envelope's own field order is unchanged, only its embedded `Region` grows.

Every embedded leaf keeps this chapter's framing (a `CanonicalF64` is a u32 LE length || 8 bytes = 12; a bool is one bare byte).

### 9.4.5 Length-prefix unification (partial)

Regime (b) (Section 2.1) is *narrowed*, not fully retired, in major 1. The `ResolvedLayoutIR` output — an independent, non-canonical `LayoutCache` chunk — moves from u64 to u32 LE counts and length prefixes, matching regime (a), and its golden anchors re-lock in the same step. Because it is non-canonical, a major-1 reader regenerates a major-0 resolved-layout cache rather than decoding its u64 form — no frozen u64 decoder is required.

The extension-declaration and edit-barrier blobs (Chapter 8) **stay regime (b)**: they are carried opaquely inside the canonical manifest (Section 7.5), which stays major 0, so a major-0 reader must keep parsing them as u64. Unifying them would require bumping the manifest itself, which this data-model bump deliberately avoids; that unification waits for a manifest-revising major.

### 9.4.6 Migration from major 0

The v0→v1 translation is total and default-filling: every new field has a canonical default, so no value is unrecoverable — including the `CreateRegion` operation payload, whose embed-

ded `Region` default-fills structurally. This is unlike the *semantic*, context-dependent operation-payload migration of Section 6.7 (which could be irreversible); the major-1 migration needs no score context.

| major-0 form                                                        | major-1 form                                                                                                                                                                                                                                                                                 |
|---------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Canvas = regions</code>                                       | <code>append layout_defaults = the A4/8 mm default</code><br>(page $105 \times 148.5$ , margins 7.5 staff spaces)                                                                                                                                                                            |
| <code>Instrument = id, name</code>                                  | <code>append range = None</code> (presence byte 0)                                                                                                                                                                                                                                           |
| <code>Region = ... (snapshot)</code>                                | <code>append permits_spanning_slurs = 0</code> (false)                                                                                                                                                                                                                                       |
| <code>CreateRegion op payload</code>                                | default-fill the embedded <code>Region</code> 's appended field<br>( <code>permits_spanning_slurs = 0</code> ); <i>canonical</i> — a vo<br>op block migrates on read, and a block bearing a<br>v1 <code>CreateRegion</code> is major 1 (a major-0-only reader<br>opens the bundle read-only) |
| <code>ResolvedLayoutIR u64 pre-</code><br><code>fixes</code>        | non-canonical <code>LayoutCache</code> : discard and regener-<br>ate at major 1 under u32 (no in-place byte transla-<br>tion)                                                                                                                                                                |
| <code>barrier / extension-</code><br><code>declaration blobs</code> | stay regime (b) u64 (manifest-embedded, manifest<br>stays major 0); unchanged in this bump                                                                                                                                                                                                   |
| <code>canonical-base</code>                                         | unchanged, byte-identical, stays major 0                                                                                                                                                                                                                                                     |
| <code>MaterializedState</code>                                      |                                                                                                                                                                                                                                                                                              |

A reader migrates a major-0 acceleration `Score` snapshot on read (cheaper than replaying the operation log); a writer emits only major-1 forms for the changed payloads and leaves the canonical base at major 0.

## 9.5 Schema Major 2

Schema major 2 is the second data-model expansion major: it fills the truncated bodies the vo baseline deliberately staged to this companion — the cross-cutting structures (`Slur`, `Tie`, `Beam`, `Spanner`), `RepeatStructure`, `Staff` and `StaffLineConfiguration`, `Instrument`, and `ScoreMetadata` — to the shapes the core specification's Chapter 5 ratifies (schema-major-2 revision). The leaf types those shapes reference are defined in the same core revision; their wire layouts are below.

### 9.5.1 Where the changed fields reach

The major is assigned per payload type, exactly as in major 1:

- **Snapshot-only:** `Instrument` (no operation embeds one); it reaches only the acceleration full-`Score` snapshot. (`RepeatStructure` was snapshot-only when this section was first ratified; the repeat-authoring pair moved it to the canonical operation layer — next bullet.)
- **Canonical operation layer:** eight operation payloads embed filled values. `CreateRepeatStructure` (this revision — Operation Catalog REPEAT STRUCTURES, discriminant 28) embeds the v2 `RepeatStructure` layout unconditionally (`kind` and `voltas` are not optional fields) and is *born at v2*: no lower-major layout for its payload exists, so every block carrying one stamps major 2 under minimal stamping. Its sibling

`DeleteRepeatStructure` (29) carries a bare identifier — a major-0 layout — so minimal stamping gives its blocks major 0; the kind discriminant itself is an append-only schema-*minor* vocabulary event (the same append mechanism the Phase-3 tranche used for 24–27 — a mechanism claim only: what a kind’s blocks *stamp* always follows minimal stamping over its payload, as discriminant 24’s always-major-2 `CreateStaff` shows). Neither changes anything about lower-major layouts. The seven payloads of the first ratification embed filled values — `CreateCrossCutting` and `ModifyCrossCutting` (a `CrossCuttingValue` embedding `Slur`, `Tie`, `Beam`, or `Spanner`); `CreateStaff` (a `Staff`, which also embeds the filled `StaffLineConfiguration`); `CreateStaffInstance` (a `StaffInstance`, whose `staff_lines_override` embeds it); `SetStaffLayout` (whose *own* `staff_lines_override` field — an op-payload field, not part of a `StaffInstance` — embeds it); `SetMetadata` (a `ScoreMetadata`); and `CreateRegion` *transitively* — its embedded `Region` may carry `StaffInstances` (`content.staff_instances`), each embedding the filled `StaffLineConfiguration`. (Reduction preconditions a *carried* region to be empty of typed children, but the byte form is representable and preserved in the canonical set, so the wire must define it.)

**Minimal stamping.** A writer **MUST** stamp each operation-envelope block with the *lowest* schema major whose layouts decode the block’s bytes to the encoded values. The stamp is therefore a pure function of the block’s values — two conforming writers stamp identical content identically, which the content hash requires (the schema version is in the chunk-hash preimage). This is the same principle the major-1 section applies to the canonical base: byte-identical content **MUST NOT** be re-stamped at a higher major. A payload whose higher-major encoding coincides byte-for-byte with its lower-major form takes the lower major; a hidden embedding site (an `Option` that is `None`, an empty `Vec`) contributes nothing. Concretely, the per-payload minimal majors at this revision: `CreateCrossCutting` / `ModifyCrossCutting` / `CreateStaff` / `SetMetadata`  $\Rightarrow$  always **2** (their v2 fills are mandatory appended fields); `CreateRegion`  $\Rightarrow$  **2** iff a carried staff instance’s `staff_lines_override` is `Some`, else **1** (the major-1 bool is always present); `CreateStaffInstance` / `SetStaffLayout`  $\Rightarrow$  **2** iff their `staff_lines_override` is `Some`, else **0**; `CreateRepeatStructure`  $\Rightarrow$  always **2** (born at v2); `DeleteRepeatStructure`  $\Rightarrow$  **0** (a bare identifier under a minor kind append); `TransposeInterval`  $\Rightarrow$  **0** (a sequence of identifiers and two `i32`s — every constituent is a major-0 layout, so the kind append is a schema *minor*, and a `TransposeInterval` rides in a major-0 block); every other payload keeps its prior major. A block stamps the maximum over its payloads. (Operation names are the Operation Catalog’s; core Chapter 6’s enum spells the creates `InsertStaff` / `InsertStaffInstance` / `InsertRegion`.)

- **The canonical base is unchanged.** The `MaterializedState` embeds none of the filled types (its values are effects, conflicts, anomalies, object states, pitch spellings, break keys, and pending records), so it remains major 0, byte-identical across the bump — the same keystone as major 1, and a conformance test **SHOULD** assert it again.
- The manifest stays major 0 (nothing here touches it).

### 9.5.2 Cross-major reader behaviour

The major-1 rules extend unchanged: non-canonical chunks at a foreign major are discarded and regenerated (or migrated on read, for the acceleration snapshot); canonical chunks parse or the bundle opens read-only. Migration *composes*: a major-2 reader migrates a major-0 snap-

shot  $v_0 \rightarrow v_1 \rightarrow v_2$  in one read, each step total and default-filling. A major-1-only reader opens a bundle read-only iff it carries an op block bearing a  $v_2$  value; a major-0-only reader additionally on  $v_1$  CreateRegion blocks, as before.

*Staging note.* As with major 1, the reference implementation raises chunk admission *per role* as the implementation phases of this revision’s push land (snapshot decode, then the canonical operation layer); the accept-set stated in Section 9.1 is the ratified end-state, not a claim that every intermediate commit already admits it.

### 9.5.3 Changed value layouts

As in major 1, **the wire form ratifies the code, not the model’s presentational field order**: a struct’s  $v_2$  layout appends its new fields *after its existing major-0/major-1 fields*, even where Chapter 5’s listing interleaves them. Concretely (every leaf under Chapter 5 regime (a) rules: LE integers, one discriminant byte per union, u32 counts and length prefixes, one presence byte per Option, one bare byte per bool):

- `Slur = id || start_event || end_event || kind || curvature_override || style.`
- `Tie = (major-0 fields) || style.`
- `Beam = id || events || level || sub_beams || geometry_override.`
- `Spanner = id || start || end || staves || kind || style` (the kind appends *after* staves; Chapter 5 lists it second, presentationally).
- `RepeatStructure = id || start || end || kind || voltas.`
- `Staff = id || name || abbreviation || instrument || default_staff_lines || group || default_clef` (appended last; Chapter 5 lists it mid-struct).
- `StaffLineConfiguration = line_count || line_spacing || line_style || bracket.` This value is embedded by Staff *and* by StaffInstance.staff\_lines\_override, which is how CreateStaffInstance, SetStaffLayout, and (transitively, through a carried region’s staff instances) CreateRegion become  $v_2$  payloads without gaining fields of their own (the CreateRegion-in-major-1 pattern).
- `Instrument = id || name || range || abbreviation || sound_config || transposition || default_clef || default_staff_lines || unpitched_members` (appended after the major-1 order id/name/range; Chapter 5’s listing order is presentational).
- `ScoreMetadata = title || composer || copyright || subtitle || lyricist || arranger || creation_timestamp || modification_timestamp || additional` (appended after the major-0 order).

### 9.5.4 New leaf-type layouts

- `SlurKind`: one byte — 0 Legato, 1 Phrase, 2 Articulation, 3 Editorial.
- `CurveDirection`: one byte — 0 Above, 1 Below.
- `CurvatureOverride = direction (Option<CurveDirection>) || height (Option<SpaceUnit>).`
- `LineStyle`: one byte — 0 Solid, 1 Dashed, 2 Dotted.
- `SpanStyle` (one shared record: `Slur.style`, `Tie.style`, `Spanner.style`) = `line (LineStyle) || thickness (Option<SpaceUnit>).`

- SubBeam = level (one byte) || events (u32 count || EventId *leaves*, each u32-length-framed 16 canonical bytes = 20 bytes — this chapter’s leaf framing, exactly as Beam.events encodes today).
- BeamGeometryOverride = slope (Option<CanonicalF64>; dimensionless) || offset (Option<SpaceUnit>).
- SpannerKind: discriminant byte || payload — 0 Generic; 1 Hairpin || HairpinDirection (one byte: 0 Crescendo, 1 Diminuendo); 2 OctaveLine || OctaveOffset (one i8); 3 PedalLine || PedalKind (one byte: 0 Sustain, 1 Sostenuto, 2 UnaCorda); 4 TrillExtension; 5 Glissando; 6 Portamento; 7 TextLine || TextLineDefinition (= one length-prefixed raw-UTF-8 string); 8 Bracket || BracketKind (one byte: 0 Square).
- RepeatKind: discriminant byte || payload — 0 SimpleRepeat || count (u32 LE); 1 Da-Capo || end\_target (TimeAnchor); 2 DalSegno || segno || end\_target; 3 Volta.
- Volta = endings (u32 count || bare u32 LE values — fixed-width primitives are unframed) || start || end (TimeAnchorS).
- SpaceUnit = one CanonicalF64, leaf-framed (12 bytes) — byte-identical to how CanonicalF64 itself encodes; the newtype adds no bytes.
- StaffBracketKind: one byte — 0 Brace, 1 Bracket.
- SoundConfiguration = u32 length || raw bytes (opaque; never interpreted by the core).
- TranspositionInterval = diatonic\_steps (i32 LE) || chromatic\_steps (i32 LE).
- UnpitchedMember = member (bare u32 LE — UnpitchedMemberId is an instrument-scoped small value, *not* a 128-bit object identifier, so the identifier-leaf framing does not apply; this matches how UnpitchedEvent.instrument\_member has encoded since major 0) || name (string) || staff\_position (bare i16 LE, likewise).
- Timestamp = one i64 LE (nanoseconds since the Unix epoch, UTC).
- MetadataEntry = key (string) || value; MetadataValue: discriminant byte || payload — 0 Text || string; 1 Integer || i64 LE; 2 Flag || bool.
- Clef, ClefShape, KeySignature, ClefChange, KeySignatureChange are *not new wire forms*: they have carried major-0 layouts since the visible-slice revision (empty sequences left Score bytes unchanged). Major 2 only *embeds* Clef in two more places (Staff.default\_clef, Instrument.default\_clef). Its layout, for the record: shape (one byte: 0 G, 1 F, 2 C, 3 Percussion) || line (i8) || octave\_shift (i8); KeySignature = fifths (i8; decoders **MUST** reject |fifths| > 7).

As in major 1: every embedded *leaf* keeps this chapter’s framing (an identifier is a u32 LE length || its 16 canonical bytes = 20; a CanonicalF64 is a u32 LE length || 8 bytes = 12); fixed-width primitive integers and bools inside structs are bare; every Option is one presence byte then the payload when present.

### 9.5.5 Migration from major 1

Total and default-filling, needing no score context; composes after the major-1 migration for major-0 input. The defaults reproduce today’s rendering exactly: a migrated document draws byte-identically.

| major-1 form                                                                                 | major-2 form                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Slur                                                                                         | <code>append kind = Legato, curvature_override = None, style = (Solid, None)</code>                                                                                                                                                                                              |
| Tie                                                                                          | <code>append style = (Solid, None)</code>                                                                                                                                                                                                                                        |
| Beam                                                                                         | <code>append sub_beams = empty, geometry_override = None</code>                                                                                                                                                                                                                  |
| Spanner                                                                                      | <code>append kind = Generic, style = (Solid, None)</code>                                                                                                                                                                                                                        |
| RepeatStructure                                                                              | <code>append kind = SimpleRepeat {count = 2} (a v1 repeat meant a repeat), voltas = empty</code>                                                                                                                                                                                 |
| Staff                                                                                        | <code>append default_clef = treble (G, line 2, shift 0)</code>                                                                                                                                                                                                                   |
| StaffLineConfiguration                                                                       | <code>append line_spacing = 1.0, line_style = Solid, bracket = None — at every embedding site: Staff.default_staff_lines, Instrument.default_staff_lines (v2-new), StaffInstance.staff_lines_override, and the SetStaffLayout op payload’s own staff_lines_override field</code> |
| Instrument                                                                                   | <code>append abbreviation = None, sound_config = empty, transposition = None, default_clef = treble, default_staff_lines = the complete v2 default (line_count = 5, line_spacing = 1.0, line_style = Solid, bracket = None), unpitched_members = empty</code>                    |
| ScoreMetadata                                                                                | <code>append subtitle / lyricist / arranger = None, both timestamps = 0 (the unset convention), additional = empty</code>                                                                                                                                                        |
| the seven embedding op payloads (incl. CreateRegion’s transitively embedded staff instances) | default-fill the embedded value(s) as above; <i>canonical</i> — a v0/v1 op block migrates on read, and a block bearing any v2-dependent bytes is major 2 (a lower-major-only reader opens the bundle read-only)                                                                  |
| canonical-base                                                                               | unchanged, byte-identical, stays major 0                                                                                                                                                                                                                                         |
| MaterializedState                                                                            |                                                                                                                                                                                                                                                                                  |

# Non-Canonical Pinned Encodings

---

**Non-canonical** means: never part of document identity. Nothing in this chapter enters a content hash that canonical state depends on, and all of it can be discarded and rebuilt without changing the document. The encodings are pinned for *reproducibility* — byte-equal conformance claims and cache correctness — not for durability.

## 10.1 LayoutObjectId (MUSCLOID)

Governed by core requirement `req:layoutir:object-id-derivation`. `LayoutObjectId = trunc128(BLAKE3(MUSCLOID || inputs))`, with three preimage shapes:

**stable** *inputs* = the source `TypedObjectId`’s canonical bytes.

**manifestation** *inputs* = source canonical bytes || `RegionId` canonical bytes (16 BE) — distinct regions give one source distinct manifestation ids.

**synthesized** *inputs* = source canonical bytes || five `u64` LE words: the `SynthesisKind` discriminant, the registry id’s high and low 64 bits (zero unless `Registered`), and the instance key’s high and low 64 bits.

`SynthesisKind` discriminants: 0 CancellationAccidental, 1 KeySignatureNatural, 2 GeneratedRest, 3 EngravedBreak, 4 MultimeasureRest, 5 Cautionary, 6 Registered.

## 10.2 ResolvedLayoutIR Canonical Output

The resolved layout’s canonical bytes are the **byte-equal conformance surface** of core Chapter 7: two conforming engravers given the same score, profile, and glyph catalog must produce identical bytes. In schema major 0 the encoding is regime (b) (`u64` LE counts and length prefixes); schema major 1 moves it to regime (a)’s `u32` prefixes with the rest of the unification (Section 9.4), re-locking this surface’s goldens. Because the surface is non-canonical, a reader regenerates a foreign-major layout cache rather than decoding it. Either way every geometric coordinate is quantized to the 1/1024 staff-space grid as a `QuantizedCoord` (8 LE bytes; Chapter 4); a non-finite or out-of-range coordinate is a determinism violation and **MUST** be rejected, never normalized. The top-level section order is: source score version || pages ||

glyphs || strokes || engraving decisions || glyph-catalog identity; within each section, elements carry their provenance (built on `MUSCLOID` ids) and their quantized geometry. The full leaf grammar is pinned by the reference implementation’s golden and round-trip anchors (Chapter 11) rather than restated field-by-field here, because the surface is non-canonical and evolves with the engraving feature set.

### 10.3 Layout Caches and Font Metrics

`LayoutCache` chunks (`ChunkKind 6`) hold cached layout artifacts; they are **always discardable**, and a reader **MUST NOT** treat a missing, stale, or undecodable layout cache as bundle corruption. The glyph-metrics identity hash is domain-separated under `MUSCFNTM` (a reserved built-in, non-canonical tag): it names the metrics a layout was computed against, so caches and conformance runs can detect font drift; it never enters canonical state.

# Golden Anchor Registry

The conformance contract binding this document to the reference implementation. Three lock classes:

**literal-byte** the test asserts exact expected bytes or exact discriminant literals — the strongest lock;

**round-trip** the test asserts `decode ◦ encode = id` and re-encode byte-stability over a corpus;

**canonical-equality** the test asserts that semantically equal values encode identically (order-independence, reduction identities).

The first block restates the anchors already recorded in the core specification’s `app:bytes` REFERENCE-IMPLEMENTATION LOCKS table (which governs them); the second block adds the anchors this document introduces, which `app:bytes` predates.

Each anchor names its crate file, then the test on a second line.

| Layout                                       | Anchor                                                                                                     | Class              |
|----------------------------------------------|------------------------------------------------------------------------------------------------------------|--------------------|
| <i>Imported from <code>app:bytes</code>:</i> |                                                                                                            |                    |
| <code>TypedObjectId</code>                   | <code>epiphany-core/src/ids.rs</code><br><code>::typed_object_id_byte_form_is_locked</code>                | literal-byte       |
| <code>Promoted VoiceId</code>                | <code>epiphany-core/src/graph.rs</code><br><code>::promoted_voice_id_byte_form_is_locked</code>            | literal-byte       |
| <code>System PitchId</code>                  | <code>epiphany-core/src/pitch.rs</code><br><code>::system_pitch_id_byte_form_is_locked</code>              | literal-byte       |
| <code>IntegrityAnomalyId</code>              | <code>epiphany-ops/src/anomaly.rs</code><br><code>::integrity_anomaly_id_byte_form_is_locked</code>        | literal-byte       |
| <code>ObjectKind (ops)</code>                | <code>epiphany-ops/src/support.rs</code><br><code>::object_kind_discriminants_are_golden</code>            | literal-byte       |
| <code>ChunkKind</code>                       | <code>epiphany-bundle/src/chunk.rs</code><br><code>::chunk_kind_discriminants_are_golden</code>            | literal-byte       |
| <code>CompressionAlgorithm</code>            | <code>epiphany-bundle/src/chunk.rs</code><br><code>::compression_algorithm_encoding_is_golden</code>       | literal-byte       |
| <code>ProfileId</code>                       | <code>epiphany-bundle/src/superblock.rs</code><br><code>::profile_id_discriminants_are_golden</code>       | literal-byte       |
| <code>ManifestId</code>                      | <code>epiphany-bundle/src/ids.rs</code><br><code>::manifest_id_is_content_derived_and_deterministic</code> | canonical-equality |
| <code>TransactionCategory</code>             | <code>epiphany-ops/src/payload.rs</code><br><code>::transaction_category_discriminants_are_golden</code>   | literal-byte       |

| Layout                                | Anchor                                                                                                                                                                    | Class                           |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| ResolutionAction                      | epiphany-ops/src/conflict.rs<br>::resolution_action_discriminants_are_golden                                                                                              | literal-byte                    |
| BlobId                                | epiphany-determinism/src/hash.rs<br>ContentHash::of_blob tests                                                                                                            | round-trip                      |
| RationalTime                          | epiphany-core/src/time.rs<br>::equal_rationals_encode_identically                                                                                                         | canonical-equality              |
| MUSCLOUD derivation                   | epiphany-layout-ir/src/provenance.rs<br>::stable_id_uses_the_ratified_muscloid_derivation                                                                                 | literal-byte                    |
| Domain-tag spellings                  | epiphany-determinism/src/domain.rs<br>::exact_tag_spellings_match_spec                                                                                                    | literal-byte                    |
| <i>Added by this document:</i>        |                                                                                                                                                                           |                                 |
| OperationKind discriminants (§6.3)    | epiphany-ops/src/payload.rs<br>::operation_kind_wire_discriminants_are_golden                                                                                             | literal-byte                    |
| OperationPayload discriminants (§6.2) | epiphany-ops/src/payload.rs<br>::operation_payload_discriminants_are_golden                                                                                               | literal-byte                    |
| OperationKindTag (§6.4)               | epiphany-ops/src/payload.rs<br>::operation_kind_tag_decode_mirrors_encode_exactly                                                                                         | round-trip                      |
| ResolveEquivocation payload (§6.2)    | epiphany-ops/src/payload.rs<br>::resolve_equivocation_payload_encodes_target_then_hash                                                                                    | literal-byte                    |
| OperationStamp (§6.5)                 | epiphany-ops/src/stamp.rs<br>::stamp_encode_is_stable                                                                                                                     | literal-byte                    |
| CausalContext (§6.5)                  | epiphany-ops/src/causal.rs<br>::canonical_encoding_is_build_order_independent                                                                                             | canonical-equality              |
| MaterializedState (§6.6.1)            | epiphany-ops/src/decode.rs<br>::reduced_states_decode_and_reencode                                                                                                        | round-trip                      |
| Operation-index payload (§7.6)        | epiphany-bundle/src/opindex.rs<br>::payload_encoding_is_golden                                                                                                            | literal-byte                    |
| Fixed header (§7.1)                   | epiphany-bundle/src/header.rs<br>::header_round_trips                                                                                                                     | round-trip                      |
| Superblock (§7.2)                     | epiphany-bundle/src/superblock.rs<br>::superblock_round_trips_through_256_bytes                                                                                           | round-trip                      |
| Chunk hash preimage (§7.3)            | epiphany-bundle/src/chunk.rs<br>::hash_preimage_matches_the_spec_layout                                                                                                   | literal-byte                    |
| ChunkRef (§7.3.1)                     | epiphany-bundle/src/chunk.rs<br>::chunk_ref_round_trips                                                                                                                   | round-trip                      |
| Manifest body (§7.5)                  | epiphany-bundle/src/manifest.rs<br>::re_encode_is_byte_identical, ::manifest_round_trips, ::duplicate_roots_collapse_on_encode, ::semver_orders_numerically_not_byte_wise | round-trip + canonical-equality |
| Barrier kinds blob (§8.1)             | epiphany-layout-ir/src/barrier.rs<br>::affected_object_kinds_blob_bytes_are_golden                                                                                        | literal-byte                    |
| Barriers blob (§8.2)                  | epiphany-layout-ir/src/barrier.rs<br>::edit_barriers_blob_bytes_are_golden                                                                                                | literal-byte                    |
| Barrier variants (§8.2)               | epiphany-layout-ir/src/barrier.rs<br>::every_scope_and_condition_variant_round_trips_byte_identically                                                                     | round-trip                      |
| Whole-Score codec (§5.1)              | epiphany-core/src/codec.rs<br>::generator_scores_round_trip, ::trailing_and_truncated_bytes_are_rejected                                                                  | round-trip                      |

| Layout                        | Anchor                                                                           | Class      |
|-------------------------------|----------------------------------------------------------------------------------|------------|
| CanonicalValue seam<br>(§5.5) | epiphany-core/src/codec.rs<br>::value_types_round_trip_over_generator_<br>corpus | round-trip |

Struct *bodies* in Chapter 5 are round-trip-locked rather than literal-byte-locked: their byte identity follows from the frozen positional rule plus the literal-byte locks on every discriminant and leaf they embed.

## 11.1 The Decode Vector Corpus

The cross-implementation decoder test is **delivered**, as a corpus of literal byte strings each paired with the verdict a decoder owes it. The reference implementation carries it at `spec/vectors/decode_vectors.txt` and regenerates it from a committed generator, so a change to any wire form moves the file.

Each line is `surface verdict class name hex`. The verdict is `accept` or `reject`. The class names *why* a rejected vector is rejected and is **informative only**: implementations need not agree on an error taxonomy.

### REQUIREMENT 11.1

A conforming decoder **MUST** accept every `accept` vector for a surface it implements, and **MUST** reject every `reject` vector for that surface. A value decoded from an `accept` vector **MUST** re-encode to exactly that vector's bytes.

Accepting a `reject` vector and then normalizing it is *accepting* it, and does not satisfy this requirement. Canonical decode is injective: distinct byte strings denote distinct values, which is what content-addressing rests on.

### RATIONALE

The corpus is not a restatement of this document; it is what a second implementation can be run against. Every rejection class in it is one the reference implementation shipped a bug in, or one whose check no injectivity fuzzer can see: a whole-value re-encode guard catches the fields a decoder *normalizes* and is blind to order-preserving sequence fields, which need per-site order checks; and a guard on an outer value can *mask* a lenient inner codec rather than fix it (Push 5).

A future revision should extend the corpus to the representative struct layouts of Section 5.4, which remain round-trip locked rather than literal-byte locked.

# Revision History

| Date      | Section                  | Change                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| July 2026 | 21, All                  | 0.1.0 — Initial companion: ratifies the previously provisional whole-Score value codec (core P11-4), operation-layer wire forms (ops), bundle physical layout (bundle P11-D2/D4/D5), operation-index payload (P12-D1), and extension-blob/edit-barrier byte forms (P12-E1/E2/E3); pins the no-varint rule, the frozen-positional schema-evolution keystone, and the id-leads envelope property; SnapshotId derivation deferred (open question).                                                                                                                                                                      |
| July 2026 | 21, Operation wire forms | 0.2.0 — Phase-3 first tranche: appended OperationKind wire discriminants 24–27 (CreateStaff, SetTimeSignature, SetTempoSegment, SetStaffLayout) with their payload layouts, the matching OperationKindTag discriminants 24–27 (kind-to-tag naming gains CreateStaff → InsertStaff), and PreconditionFailureReason 11 (TempoMapMalformed) — a schema- <i>minor</i> evolution under this document’s own append-only rules (Chapter 9); no existing assignment changed. Semantics: Operation Catalog 0.5.0 (CREATESTAFF, METER AND TEMPO OVERWRITES, SETSTAFFLAYOUT, and the value-restoring UNDOTRANSACTION revision). |

| Date      | Section                                    | Change                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| July 2026 | 21, Schema evolution / Graph value layouts | 0.3.0 — Defines <b>schema major 1</b> , the first data-model expansion major (Section 9.4): appends <code>Canvas.layout_defaults</code> (new <code>CanvasLayoutDefaults/CanvasSize/CanvasMargins</code> layouts, <code>snapshot-only</code> ), <code>Instrument.range</code> ( <code>Option&lt;PitchRange&gt;</code> , <code>snapshot-only</code> ), and <code>Region.permits_spanning_slurs</code> (also inside the <i>canonical</i> <code>CreateRegion</code> operation payload); narrows regime (b) by unifying the non-canonical resolved-layout length prefixes to <code>u32</code> (the manifest-embedded barrier blobs stay <code>u64</code> , partly resolving the standing open question); pins per-payload-type major assignment (the canonical-base <code>MaterializedState</code> stays major 0, byte-identical), the accept-set gate ( <code>[MIN,MAX]</code> ), the cross-major reader rules (discard-and-regenerate non-canonical chunks; parse-or-read-only for canonical ones, so a major-0 reader opens a bundle with <code>v1 CreateRegion ops read-only</code> ), and the total default-filling <code>v0→v1</code> migration table. Clarifies that any field add is a major change regardless of Option-ness. |
| July 2026 | 21, Effect vocabularies / Extension blobs  | 0.4.0 — Pass-12 G-ratification: appends <code>PreconditionFailureReason</code> 12 ( <code>SystemDerivedContentImmutable</code> , <code>P12-K3</code> ) and 13 ( <code>RecreateContentMismatch</code> , <code>P12-K9</code> ), and <code>ReanchorReason</code> 6 ( <code>SameCanvasNearer</code> , <code>P12-C4</code> ) — <i>schema-minor</i> appends under Chapter 9; no existing assignment changed. Adds the unsafe-edit tombstone <i>encoding</i> open question (from <code>P12-E5</code> ; semantics ratified in core Chapter 8). Semantics: Operation Catalog 0.6.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Date      | Section                                    | Change                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| July 2026 | 21, Schema evolution / Graph value layouts | 0.5.0 — Defines <b>schema major 2</b> , the second data-model expansion major (Section 9.5): fills the staged truncations — <code>Slur</code> (kind / curvature override / style), <code>Tie</code> (style), <code>Beam</code> (sub-beams / geometry override), <code>Spanner</code> (kind, with the leading <code>Generic</code> migration default / style), <code>RepeatStructure</code> (kind / voltas), <code>Staff</code> (default_clef), <code>StaffLineConfiguration</code> (spacing / line style / bracket), <code>Instrument</code> (six appended fields), and <code>ScoreMetadata</code> (six appended fields incl. the strictly-authored timestamps) — with layouts for the new leaf types and the total default-filling v1→v2 migration table. Assigns the major per payload type: <code>snapshot-only</code> for <code>Instrument</code> / <code>RepeatStructure</code> ; <i>canonical</i> for the seven embedding op payloads ( <code>Create/ModifyCrossCutting</code> , <code>CreateStaff</code> , <code>CreateStaffInstance</code> , <code>SetStaffLayout</code> , <code>SetMetadata</code> , and <code>CreateRegion</code> transitively through carried staff instances); the canonical base and manifest stay major 0, byte-identical. <code>Accept-set</code> widens to $\{0, 1, 2\}$ (staged per role as the implementation phases land). Semantics: core specification schema-major-2 revision (leaf-type definitions, clef/key content-model ratification, <code>ClefId</code> retirement, strictly-authored timestamp requirement). |
| July 2026 | 21, Operation vocabularies                 | 0.6.0 — Repeat authoring (schema-major-2 revision, Phase D): appends <code>OperationKind</code> 28 ( <code>CreateRepeatStructure</code> , <code>lp(RepeatStructure)</code> ), born at v2 — every block carrying one stamps major 2 under minimal stamping) and 29 ( <code>DeleteRepeatStructure</code> , bare identifier — minimal stamping over its major-0 payload layout gives major 0; the discriminant itself a <i>schema-minor</i> vocabulary append), with the matching <code>OperationKindTag</code> 28/29 (name-verbatim projection). No existing assignment changed; Section 9.5 restates the per-payload stamping with the pair. Semantics: <code>Operation Catalog REPEAT STRUCTURES</code> , 0.7.0; re-anchoring row (nearest-surviving-anchor across every anchor site, cascade when none): core Chapter 6 rule table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

| Date      | Section                    | Change                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| July 2026 | 21, Operation vocabularies | 0.7.0 — Transpose algebra (Push 4a): appends <code>OperationKind 30</code> ( <code>TransposeInterval: seq<sup>↑</sup>(PitchId)    two i32s</code> ) and the matching <code>OperationKindTag 30</code> . Every constituent is a major-0 layout, so the payload stamps major 0 and the discriminant is a schema- <i>minor</i> vocabulary append — no major 3. Introduces the <code>seq<sup>↑</sup></code> notation (strictly increasing; a decoder <b>MUST</b> reject a duplicate rather than normalize it) and marks 9 ( <code>Transpose</code> ) frozen, its <code>seq<sup>↑</sup></code> being a multiset. No existing assignment changed. Semantics: Operation Catalog <code>TRANSPOSEINTERVAL</code> , 0.8.0; algebra: core Chapter 2, <code>TRANSPOSITION AND THE INTERVAL TYPE</code> . |
| July 2026 | 21, Chunk references       | 0.8.0 — <code>StrictCompressionAlgorithm decode</code> (Push 5, P3): <code>req:binfmt:compression-none-parameter</code> <b>MUST</b> reject a non-zero parameter byte on the <code>None</code> discriminant. Ignoring it made the codec non-injective — two byte strings, one value — which every structure embedding a <code>ChunkRef</code> inherited; the operation index, having no whole-value re-encode guard, accepted both. Found by the wire-decode fuzzer. Supersedes the core specification’s “ignored on read”; no wire layout changed and no conforming writer is affected.                                                                                                                                                                                                      |
| July 2026 | 21, Golden anchors         | 0.9.0 — Ratifies the decode vector corpus (Push 5, P4): <code>req:binfmt:decode-vectors</code> and Section 11.1. The cross-implementation decoder test moves out of the <code>ABOUT THIS COMPANION</code> exclusion list and out of the <code>GOLDEN ANCHOR REGISTRY</code> ’s “deferred conformance harness” note: literal byte vectors with accept/reject verdicts now exist ( <code>spec/vectors/decode_vectors.txt</code> ), and a conforming decoder owes them the verdict. Accepting a reject vector and normalizing it is <i>accepting</i> it. The wire-format fuzzer remains an implementation deliverable. No wire layout changed.                                                                                                                                                  |