# **Pre-Coding Essentials (Component: schemas/ballot\_tally.schema.json, Version/FormulaID: VM-ENGINE v0) — 15/89**

## **1) Goal & Success**

Goal: JSON Schema for **aggregated tallies by Unit** used directly by TABULATE (no per-ballot data).

Success: Validates canonical IDs/links, enforces **exactly one** tally shape per file (plurality | approval | score | ranked\_irv | ranked\_condorcet), and checks basic tally sanity so downstream math is deterministic.

## **2) Scope**

In scope: Top-level metadata (IDs, label, reg link), one-of per ballot type, per-Unit fields, non-negativity, basic sanity: Σ(valid option tallies) + invalid\_or\_blank ≤ ballots\_cast.

Out of scope: Cross-file referential checks (unknown Unit/Option IDs), hierarchy rules, gating/threshold logic (pipeline validates).

## **3) Inputs → Outputs**

Inputs: ballot\_tally.json (aggregated counts).

Outputs: Pass/fail against schema; on pass, loader builds typed UnitTallies used to compute UnitScores and turnout.

## **4) Entities/Fields (schema shape to encode)**

**Root object**

id **(required, string)** — TLY:<name>:v<digits>

label **(required, string)** — human-readable dataset label (surfaces in reports)

reg\_id **(required, string)** — REG:<name>:<version> (must correspond to the DivisionRegistry used)

ballot\_type **(required, enum)** — plurality | approval | score | ranked\_irv | ranked\_condorcet

tallies **(required, object)** — **exactly one** of the following keys must be present:

plurality

approval

score

ranked\_irv

ranked\_condorcet

notes *(optional, string)*

**Per-type payloads (mutually exclusive)**

**plurality**

units **(required, array)** of objects:

unit\_id **(string)** — U:…

ballots\_cast **(integer ≥ 0)**

invalid\_or\_blank **(integer ≥ 0)**

votes **(object)** — map OPT:<id> → **integer ≥ 0**

**Sanity (schema-level where possible):** invalid\_or\_blank ≤ ballots\_cast  
 *(Full Σ votes ≤ ballots\_cast - invalid\_or\_blank rechecked in pipeline.)*

**approval**

units array of:

unit\_id, ballots\_cast, invalid\_or\_blank as above

approvals **(object)** — map OPT:<id> → **integer ≥ 0**

**Sanity:** same as plurality; pipeline ensures Σ approvals\_for\_all\_options ≤ ballots\_cast × max\_approvals\_per\_ballot if such a cap exists (usually unlimited).

**score**

scale\_min **(integer, default 0)**, scale\_max **(integer, > scale\_min)**

ballots\_counted **(integer ≥ 0)** — **per unit** (inside units[])

units array of:

unit\_id, ballots\_cast, invalid\_or\_blank

ballots\_counted **(integer ≥ 0)**

score\_sum **(object)** — map OPT:<id> → **integer ≥ 0**

**Sanity:** ballots\_counted ≤ ballots\_cast - invalid\_or\_blank; per-option sums unconstrained by schema beyond non-negativity; pipeline enforces bounds vs scale if needed.

**ranked\_irv**

units array of:

unit\_id, ballots\_cast, invalid\_or\_blank

ballots **(array)** of compressed rankings:

{ ranking: array<string /\* OPT:… \*/> (uniqueItems: true), count: integer ≥ 1 }

**Sanity:** Σ(count) ≤ ballots\_cast - invalid\_or\_blank.

**ranked\_condorcet**

Same shape as ranked\_irv.units[].ballots.

Lists should already be in **canonical order** (Units by unit\_id lexicographically; Options by order\_index then ID). Schema can’t enforce; loader will sort before hashing.

## **5) Variables (validators & enums used in schema)**

## **6) Functions**

(Schema only.)

## **7) Algorithm Outline (schema authoring steps)**

Use JSON Schema **2020-12**; set $id, $schema.

$defs: TlyId, RegId, UnitId, OptId, small object schemas per payload.

Root object: required: ["id","label","reg\_id","ballot\_type","tallies"], additionalProperties: false.

**One-of** selector on tallies: require exactly one of the five keys; tie ballot\_type to the present key using conditional subschemas (if/then with const).

Arrays: minItems: 0, items typed; objects are additionalProperties: false.

Integer minimum: 0 for all counts; add local comparisons where possible (invalid\_or\_blank ≤ ballots\_cast).

Leave cross-field sums (e.g., Σ votes) to pipeline validation for clarity and performance.

$comment (non-normative) documenting canonical LF/UTF-8/sorted-keys policy (enforced at I/O layer).

## **8) State Flow**

Loader: schema-validate → normalize orders → construct UnitTallies → TABULATE consumes tallies to produce UnitScores and turnout per unit.

## **9) Determinism & Numeric Rules**

Integers only; no floats in inputs.

Canonical serialization (UTF-8, LF, sorted keys) enforced outside schema; stable ID patterns aid reproducible hashing.

## **10) Edge Cases & Failure Policy**

Multiple payloads present → **schema fail**.

ballot\_type/payload mismatch → **schema fail**.

Negative counts or invalid\_or\_blank > ballots\_cast → **schema fail**.

Unknown fields anywhere → **schema fail**.

Cross-file problems (unknown Unit/Option IDs, mismatched REG) → **pipeline fail**.

## **11) Test Checklist (must pass)**

Minimal valid example for each payload type → **pass**.

File with both approval and plurality under tallies → **fail**.

ranked\_irv with duplicated option inside one ranking → **fail** (via uniqueItems).

score with scale\_max ≤ scale\_min → **fail**.

invalid\_or\_blank > ballots\_cast in any unit → **fail**.

Pipeline tests: Σ option tallies + invalid\_or\_blank ≤ ballots\_cast across all units; unknown OPT:/U: rejected with precise errors.