# **Pre-Coding Essentials (Component: vm\_pipeline/src/load.rs, Version/FormulaID: VM-ENGINE v0)**

## **1) Goal & Success**

**Goal:** Load all local inputs (Registry, Options, BallotTally, ParameterSet, optional Manifest) into a **LoadedContext** for downstream stages. No network. No semantics yet.

**Success:** Same bytes ⇒ same parsed structs across OS; canonicalization available for hashing; IDs preserved; ordering left unchanged until later stages. Canonical JSON rules (UTF-8, LF, sorted keys) used when emitting/recording canonical bytes.

## **2) Scope**

**In scope:** Read local files; deserialize JSON → engine types; optionally produce canonical bytes + SHA-256 for determinism; accept **either** explicit file paths or a **Manifest** that references inputs.

**Out of scope:** Cross-object validation (tree, magnitudes, tallies), gates/algorithms, reporting. These are later states.

## **3) Inputs → Outputs (with schemas/IDs)**

**Inputs (files):** division\_registry.json (REG / Units / Adjacency), options.json (OPT list), ballot\_tally.json (TLY), parameter\_set.json (PS), optional manifest.json. ID formats per Annex B Part 0.

**Outputs:** **LoadedContext**: { DivisionRegistry, Units, Options (with order\_index), BallotTally, ParameterSet, engine refs } for later stages.

**Canonicalization (optional):** Canonical bytes + SHA-256 over sorted-key JSON, LF, NFC strings; timestamps UTC if present.

## **4) Entities/Tables (minimal)**

## **5) Variables (only ones used here)**

**None applied by LOAD.** It just parses the **ParameterSet** map; semantics happen later. (VM-VAR ranges & defaults are normative context.)

## **6) Functions (signatures only)**

fn load\_from\_manifest(path: &Path) -> Result<LoadedContext> — resolve input refs, then call the specific loaders.

fn load\_division\_registry(path: &Path) -> Result<DivisionRegistry>

fn load\_options(path: &Path) -> Result<Vec<Option>>

fn load\_ballot\_tally(path: &Path) -> Result<BallotTally>

fn load\_parameter\_set(path: &Path) -> Result<ParameterSet>

fn load\_and\_canonicalize<T: DeserializeOwned + Serialize>(path: &Path) -> Result<(T, CanonicalBytes, Sha256)> (utility)

## **7) Algorithm Outline (bullet steps)**

If given a **Manifest**, resolve absolute file paths; else use CLI-provided paths.

For each artifact: read bytes → JSON parse → (optional) canonicalize & hash for determinism log.

Assemble **LoadedContext** with exact IDs and arrays as in inputs (do **not** re-order here).

Return **LoadedContext**; **VALIDATE** stage runs next.

## **8) State Flow (very short)**

Pipeline: **LOAD → VALIDATE → TABULATE …** (fixed order). On LOAD error, stop with a clear error.

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

**Offline only**; no network I/O.

Canonical JSON on demand: **UTF-8, LF, sorted keys; omit nulls; NFC strings**; hash with **SHA-256**.

Lists will be **sorted later** before hashing outputs (Units by ID; Options by order\_index then ID).

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

**Missing file / unreadable / non-UTF8** → typed I/O error.

**JSON parse error** → typed JSON error.

**Oversize file** (over limit) → validation-style error from loader.

**Canonical hash mismatch** (when verifying against Manifest/fixture) → explicit HashMismatch.

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

Loading all four artifacts (REG/OPT/TLY/PS) from local paths succeeds; **no network** attempted.

Canonicalization of the same JSON with shuffled keys yields **identical bytes** (+ trailing \n) and **same SHA-256** across OS.

Fixture acceptance: VM-TST-019/020 determinism relies on these rules (identical Result/RunRecord hashes across runs/OS).