# **Pre-Coding Essentials (Component: crates/vm\_io/src/lib.rs, Version/FormulaID: VM-ENGINE v0) — 30/89**

## **1) Goal & Success**

Goal: Public surface for **vm\_io** — canonical JSON I/O, schema validation, path resolution, hashing, and high-level loaders that return **typed vm\_core structs**.

Success: vm\_pipeline/vm\_cli can load **manifest → registry → params → ballots/tally (+adjacency)**, validate against schemas, produce **canonical bytes + SHA-256**, and surface precise errors. No network, no UI.

## **2) Scope**

In scope: Module exports, error types, trait re-exports, convenience loaders/writers, schema validator wiring, digest helpers.

Out of scope: Algorithms/pipeline logic, RNG, report rendering.

## **3) Inputs → Outputs**

Inputs: Local JSON files (manifest.json, division\_registry.json, parameter\_set.json, ballots.json *or* ballot\_tally.json, optional adjacency.json if split).

Outputs:

**Typed** values (DivisionRegistry, Params, …) from vm\_core.

**LoadedContext** (ephemeral bundle for pipeline).

Canonical JSON bytes + **SHA-256** digests for artifacts.

Validation errors with JSON Pointers to failing paths.

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

## **5) Variables (feature/config toggles surfaced by this lib)**

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

rust

CopyEdit

// Re-exports

pub use vm\_core::{ids::\*, entities::\*, variables::Params};

// Error model

#[derive(thiserror::Error, Debug)]

pub enum IoError {

#[error("read error: {0}")] Read(std::io::Error),

#[error("write error: {0}")] Write(std::io::Error),

#[error("json parse error at {pointer}: {msg}")] Json { pointer: String, msg: String },

#[error("schema validation failed at {pointer}: {msg}")] Schema { pointer: String, msg: String },

#[error("manifest violation: {0}")] Manifest(String),

#[error("canonicalization: {0}")] Canon(String),

#[error("hashing: {0}")] Hash(String),

#[error("path: {0}")] Path(String),

}

// Canonical JSON (sorted keys, LF)

pub mod canonical\_json {

pub fn to\_canonical\_bytes<T: serde::Serialize>(value: &T) -> Result<Vec<u8>, IoError>;

pub fn write\_canonical\_file<T: serde::Serialize, P: AsRef<Path>>(value: &T, path: P) -> Result<(), IoError>;

}

// SHA-256 digests

pub mod hasher {

pub fn sha256\_hex(bytes: &[u8]) -> String;

pub fn sha256\_file<P: AsRef<Path>>(path: P) -> Result<String, IoError>;

}

// Manifest & path resolution

pub mod manifest {

pub struct Manifest { /\* typed view of schemas/paths/expect \*/ }

pub fn load\_manifest<P: AsRef<Path>>(path: P) -> Result<Manifest, IoError>;

pub fn resolve\_paths(base: &Path, man: &Manifest) -> Result<ResolvedPaths, IoError>;

}

// JSON Schema validation helpers

pub mod schema {

pub enum SchemaKind { DivisionRegistry, ParameterSet, Ballots, BallotTally, Manifest, Result, RunRecord, FrontierMap }

pub fn validate\_value(kind: SchemaKind, value: &serde\_json::Value) -> Result<(), IoError>;

}

// High-level loaders (return vm\_core types)

pub mod loader {

pub struct LoadedContext {

pub reg: DivisionRegistry,

pub params: Params,

pub tally\_or\_ballots: TallyOrBallots,

pub adjacency\_inline: Option<Vec<Adjacency>>, // if not separate

pub ids: LoadedIds, // echo of REG/TLY/PS

}

pub enum TallyOrBallots {

Ballots(BallotsRaw), // typed in vm\_io

Tally(UnitTallies), // typed in vm\_io

}

pub fn load\_all\_from\_manifest<P: AsRef<Path>>(path: P) -> Result<LoadedContext, IoError>;

pub fn load\_registry<P: AsRef<Path>>(path: P) -> Result<DivisionRegistry, IoError>;

pub fn load\_params<P: AsRef<Path>>(path: P) -> Result<Params, IoError>;

pub fn load\_ballots<P: AsRef<Path>>(path: P) -> Result<BallotsRaw, IoError>;

pub fn load\_tally<P: AsRef<Path>>(path: P) -> Result<UnitTallies, IoError>;

}

## **7) Algorithm Outline (module layout)**

**canonical\_json**

Serialize via serde\_json::Serializer with **stable key order** (pre-sort BTreeMap/custom map walker).

Force LF endings; UTF-8; no trailing spaces; optionally ensure numeric types emitted as integers.

**hasher**

sha256\_hex over **canonical bytes** only; file variant reads in chunks (no mmap requirement).

**manifest**

Load JSON → schema-validate → reject URLs → resolve relative paths against manifest directory → return ResolvedPaths.

Optional “expect” check (FormulaID/engine version) performed here and errors early.

**schema**

Load static JSON Schemas (bundled at compile time or read from schemas/) → validate values → map first failure to IoError::Schema with JSON Pointer.

**loader**

load\_all\_from\_manifest: orchestrates full load; enforces **exactly one** of ballots/tally; returns LoadedContext.

When loading raw ballots/tallies, normalize **option and unit ordering** (stable sorts) before handing to pipeline.

## **8) State Flow**

CLI/pipeline calls load\_all\_from\_manifest → gets LoadedContext → pipeline executes **VALIDATE → TABULATE → …** using typed data; vm\_io later writes **Result/RunRecord/FrontierMap** via canonical writer and hashes.

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

Canonical JSON: **sorted keys**, **LF**, **UTF-8**, **UTC** timestamps (where present).

Use BTreeMap or explicit sort before serialization.

**No floats** introduced; counts/ratios remain integers until report layer.

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

Paths that start with http:// or https:// → **reject**.

Relative path traversal (..) allowed at schema level but may be rejected by policy here if it escapes the workspace root.

Oversized file or excessive nesting → **fail** with clear limit names (io.max\_bytes, io.max\_depth).

Both ballots\_path and ballot\_tally\_path present or both absent → **fail**.

Schema disabled feature: if io.schema.enabled=0, still parse but emit a **warning** field in IoError type isn’t appropriate; instead return Ok and rely on pipeline validation (documented).

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

Canonical writer: serializing the same structure twice yields **byte-identical** output; keys sorted; LF enforced.

Hashing: sha256\_file of a file equals sha256\_hex(to\_canonical\_bytes(parsed)) for canonical sources.

Manifest: URL paths **rejected**; exactly-one ballots/tally enforced; expectations mismatch triggers error.

Schema: invalid registry/tally/params fail with precise JSON Pointer.

Loader: happy paths for **raw ballots** and **tally**; option/unit lists normalized deterministically.

DoS guards: files > limit and depth > limit both fail fast with clear messages.