# **Pre-Coding Essentials (Component: crates/vm\_io/src/manifest.rs, Version/FormulaID: VM-ENGINE v0) — 32/89**

## **1) Goal & Success**

Goal: Parse, validate, and resolve the **run manifest** into concrete, local file paths and expectations for a deterministic offline run.

Success: Given a manifest.json, return a typed Manifest + ResolvedPaths with **exactly one** ballots source selected, **no URLs**, paths resolved against the manifest’s directory, and (if present) **expectations/digests** verified.

## **2) Scope**

In scope: JSON parse → schema check → typed struct; relative-path resolution; “one-of” ballots vs tally; optional expect{formula\_id, engine\_version} check; optional digests verification; precise error mapping.

Out of scope: Reading the target files (done in loader.rs), hashing bytes (in hasher.rs), canonical JSON writing.

## **3) Inputs → Outputs**

Inputs: Path to manifest.json on disk.

Outputs:

Manifest (typed view of fields).

ResolvedPaths (absolute or base-relative normalized paths for registry, params, ballots **or** tally, optional adjacency).

Optional checks run: expectations verified; digests verified (if provided).

## **4) Entities/Tables**

## **5) Variables (module knobs)**

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

rust

CopyEdit

// Public API

pub struct Manifest {

pub id: String, // MAN:…

pub reg\_path: String,

pub params\_path: String,

pub ballots\_path: Option<String>,

pub ballot\_tally\_path: Option<String>,

pub adjacency\_path: Option<String>,

pub expect: Option<Expect>,

pub digests: Option<BTreeMap<String, DigestEntry>>,

}

pub struct Expect {

pub formula\_id: Option<String>,

pub engine\_version: Option<String>,

}

pub struct DigestEntry { pub sha256: String }

pub enum BallotSource { Ballots, Tally }

pub struct ResolvedPaths {

pub base\_dir: camino::Utf8PathBuf,

pub reg: camino::Utf8PathBuf,

pub params: camino::Utf8PathBuf,

pub ballots: Option<camino::Utf8PathBuf>,

pub tally: Option<camino::Utf8PathBuf>,

pub adjacency: Option<camino::Utf8PathBuf>,

pub source: BallotSource,

}

// Top-level

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

pub fn validate\_manifest(man: &Manifest) -> Result<(), IoError>;

pub fn resolve\_paths<P: AsRef<std::path::Path>>(manifest\_file: P, man: &Manifest)

-> Result<ResolvedPaths, IoError>;

pub fn enforce\_expectations(man: &Manifest, engine\_version: &str, formula\_id\_hex: &str)

-> Result<(), IoError>;

pub fn verify\_digests(paths: &ResolvedPaths, digests: &BTreeMap<String, DigestEntry>)

-> Result<(), IoError>;

## **7) Algorithm Outline (implementation plan)**

**Read & parse**

Read file (cap mf.max\_bytes).

Parse into serde\_json::Value; map parse errors to IoError::Json { pointer, msg } (pointer “/” if not available).

**Schema validation**

Validate against schemas/manifest.schema.json when mf.strict\_schema.

Fail with IoError::Schema on first violation (carry JSON Pointer).

**To typed Manifest**

Deserialize to the Manifest struct; additionalProperties: false enforced by schema, not by struct.

Quick surface checks:

**Exactly one** of ballots\_path or ballot\_tally\_path is Some.

All path strings **do not** start with http:// or https:// when mf.reject\_urls.

**Resolve paths**

base\_dir = manifest\_file.parent().unwrap\_or(".").

For each present path, join to base\_dir using camino::Utf8PathBuf, then **normalize** (.normalize() or manual dot-segment removal).

If mf.allow\_parent\_traversal == 0, reject any resolved path that escapes base\_dir after normalization.

Do **not** require files to exist here (loader does that), but you may optionally metadata to give earlier errors.

**Decide source**

Set BallotSource::Ballots if ballots\_path.is\_some() else Tally.

**Expectations (optional)**

If expect.formula\_id present, compare to provided formula\_id\_hex (case-insensitive hex compare).

If expect.engine\_version present, exact string compare.

Mismatch → IoError::Manifest("expectation mismatch: …").

**Digests (optional)**

If digests present and mf.verify\_digests, compute SHA-256 for each listed path **as written in the manifest** (relative to base). Compare hex (case-insensitive).

Any mismatch → IoError::Manifest("digest mismatch for <path>").

**Return**

On success, return Manifest + ResolvedPaths (or ResolvedPaths only if the loader calls resolve\_paths directly).

## **8) State Flow**

vm\_cli/vm\_pipeline calls:

load\_manifest → validate\_manifest.

resolve\_paths → enforce\_expectations (with engine/formula data).

hand ResolvedPaths to loader.rs to actually read Registry/Params and Ballots **or** Tally.

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

Determinism supported by forcing **local files** and optional digest checks prior to execution.

No numeric operations here.

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

Both ballots\_path **and** ballot\_tally\_path present (or neither) → **error**.

Any \*\_path begins with http(s):// → **error**.

Path normalization escapes base when traversal disallowed → **error**.

digests map includes a path not present in the manifest → ignore or warn? **Choose error** for strictness.

Hex digest not 64-hex → **error**.

expect provided but engine/formula not passed to enforce\_expectations → caller bug; document in API contract.

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

**Happy (tally):** reg\_path, params\_path, ballot\_tally\_path, no URLs → parse/validate/resolve succeeds; source=Tally.

**Happy (ballots):** same but with ballots\_path; source=Ballots.

**Both/None ballot sources:** validate\_manifest fails with clear message.

**URL rejection:** any http(s)://… in paths → fail.

**Traversal:** with mf.allow\_parent\_traversal=0, path ../outside.json rejects; with it on, resolving succeeds.

**Expectations:** mismatch in formula\_id or engine\_version → fail; match → pass.

**Digests:** correct hex passes; wrong hex or mismatched file content → fail; non-hex → fail.

**Determinism:** resolving the same manifest on different OS yields identical normalized Utf8PathBuf strings (relative to base).