# **Pre-Coding Essentials (Component: crates/vm\_io/src/hasher.rs, Version/FormulaID: VM-ENGINE v0) — 33/89**

## **1) Goal & Success**

Goal: Provide **canonical hashing utilities** (SHA-256) used for digests and IDs, fed strictly with **canonical JSON bytes**.

Success: Same struct → same canonical bytes → same hex digest across OS/arch; supports streaming file hashes; exposes helpers for Result/RunRecord IDs and **Formula ID** (FID) from the Normative Manifest.

## **2) Scope**

In scope: SHA-256 over bytes/reader, helpers to hash **canonical JSON** values, FID builder (hash over NM fields only), lowercase hex encoding.

Out of scope: JSON canonicalization itself (lives in canonical\_json.rs), file I/O policy, report formatting.

## **3) Inputs → Outputs**

Inputs: &[u8], Read streams, or serde::Serialize values that have been canonicalized.

Outputs: Lowercase **hex** digests (String), plus tiny wrappers for ID strings (RES:…, RUN:…, FR:… computed elsewhere from digests).

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

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

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

rust

CopyEdit

use std::io::Read;

/// Hash raw bytes → lowercase hex.

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

/// Hash a reader (stream) → lowercase hex (no mmap).

pub fn sha256\_stream<R: Read>(mut r: R) -> Result<String, IoError>;

/// Canonicalize then hash a serializable value.

pub fn sha256\_of\_canonical<T: serde::Serialize>(value: &T) -> Result<String, IoError>;

/// Build a short hash token (e.g., first 12–16 hex chars) for IDs.

pub fn short\_hex(full\_hex: &str, len: usize) -> Result<String, IoError>;

/// Compute Formula ID from a Normative Manifest (NM) value:

/// \*drop\* origin fields; sort keys; hash canonical bytes.

pub fn formula\_id\_from\_nm(nm: &serde\_json::Value) -> Result<String, IoError>;

/// Convenience: digest a file at path.

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

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

**sha256\_hex**: sha2::Sha256::digest(bytes) → lowercase hex via hex::encode.

**sha256\_stream**: read in hash.buf\_size chunks; update hasher incrementally; return hex.

**sha256\_of\_canonical**: call canonical\_json::to\_canonical\_bytes(value) then sha256\_hex.

**short\_hex**: validate len>0 && len<=full.len() and all-hex; return Ok(full[..len].to\_string()).

**formula\_id\_from\_nm**:

Accept an NM as serde\_json::Value.

**Strip** non-normative fields (origin, timestamps) recursively as specified; keep only the four normative blocks (schema\_version, variables, constants, compat).

Re-serialize with canonical JSON; sha256\_hex over those bytes.

**sha256\_file**: open in binary; call sha256\_stream.

## **8) State Flow**

Writers/Builders: canonical\_json::\* → sha256\_of\_canonical → hex digest.

Manifests/Run: manifest.verify\_digests and pipeline ID builders consume sha256\_\* outputs to compare/compose IDs.

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

Determinism comes from hashing **canonical bytes only**; hex always lowercase; no locale/time/path influence.

No floats processed here; any floats should already be rejected by canonicalization layer when hashing artifacts.

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

Empty input bytes → valid hash of empty string (document value).

Non-hex passed to short\_hex → IoError::Hash("non-hex input").

NM missing required normative blocks in formula\_id\_from\_nm → IoError::Hash("incomplete NM").

Reader errors bubble as IoError::Hash with source detail.

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

**Byte equality:** same canonical struct twice → identical digest.

**Stream vs bytes:** sha256\_stream(File) equals sha256\_hex(read\_all\_bytes).

**Short hex:** length guard and hex validation; short\_hex(full, 16) stable.

**NM FID:** adding an origin block does **not** change FID; changing a variable default **does**.

**Cross-platform:** digests equal on Win/macOS/Linux for same canonical input.