# **Pre-Coding Essentials (Component: crates/vm\_io/src/canonical\_json.rs, Version/FormulaID: VM-ENGINE v0) — 31/89**

## **1) Goal & Success**

Goal: Produce **byte-identical** JSON for hashing and artifacts: UTF-8, **sorted keys**, **LF** newlines, stable escaping, and no nondeterministic whitespace.

Success: Same Rust structs → same bytes across OS/arch; maps serialized with lexicographic key order; writer enforces LF; output feeds SHA-256 and on-disk files exactly.

## **2) Scope**

In scope: Canonical serializer (to bytes / to file), recursive map key ordering, stable string escaping, optional pretty printer for human view that still preserves LF and key order.

Out of scope: Schema validation, business logic, hashing (separate module), timestamps generation.

## **3) Inputs → Outputs**

Inputs: Any serde::Serialize value (typically Result, RunRecord, FrontierMap, registries, params).

Outputs: Vec<u8> canonical bytes; or file written with those bytes.

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

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

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

rust

CopyEdit

/// Return canonical JSON bytes (UTF-8, sorted keys, compact by default).

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

/// Write canonical JSON file (creates parent dirs, atomically replace).

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

/// Pretty writer variant (indented) that still sorts keys and enforces LF.

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

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

**Key ordering (core):**

Implement a CanonicalValue transformer: visit serde\_json::Value, recursively convert all objects’ key/value pairs into a BTreeMap<String, Value> (lexicographic by bytes), leaving arrays in original order.

For direct Serialize inputs, first serialize to Value (in-memory), transform, then stream out.

**Compact writer:**

Use serde\_json::Serializer with a custom Formatter that emits **no extra spaces**, **no trailing whitespace**, and **\n** if a newline is required (e.g., after final byte only if we decide to append one—default: **no trailing newline**).

Ensure escape\_ascii is **off** so UTF-8 stays UTF-8; rely on serde’s stable escaping for control characters and quotes.

**Pretty writer (optional):**

PrettyFormatter with fixed two-space indentation; override newline to \n. Maintain sorted keys via the same CanonicalValue step.

**LF enforcement:**

When writing to disk, normalize any platform line endings the formatter might introduce (our formatter will only use \n); ensure file is opened/written in binary mode to avoid OS conversion.

**Atomic file write:**

Write to path.tmp then rename to path to avoid partial files.

**Deterministic numbers:**

We only serialize integers/ratios; do not accept f64 in public API for canonical artifacts. If encountered in a generic Value, return IoError::Canon("float not allowed").

## **8) State Flow**

vm\_pipeline prepares structs → calls to\_canonical\_bytes → hashes bytes → write\_canonical\_file to persist identical content on all platforms. Reports read these artifacts later.

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

Keys sorted lexicographically; arrays/order-sensitive sequences untouched.

UTF-8 only; **no BOM**; **LF** newlines; compact spacing fixed.

No floats permitted in canonical artifacts; integers and strings only.

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

**Non-string map keys** (rare with serde): reject with IoError::Canon("non-string key").

**Float present**: reject as above.

**Very large maps**: BTreeMap transformation is O(n log n); acceptable; streaming path stays deterministic.

**Invalid UTF-8 in strings**: impossible by serde contract; if encountered in raw bytes, treat as parse error.

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

Same struct serialized twice → **byte-identical**.

Same map with different insertion orders → **byte-identical** after sorting.

Windows/macOS/Linux produce identical bytes for the same value.

Pretty vs compact differ only in insignificant whitespace; hashes computed from **compact** form are stable.

Round-trip: parse(canonical\_bytes) → reserialize → identical.

Reject floats and non-string keys with clear IoError::Canon.