# **Pre-Coding Essentials (Component: crates/vm\_core/src/lib.rs, Version/FormulaID: VM-ENGINE v0) — 22/89**

## **1) Goal & Success**

Goal: Public surface of **vm\_core** (IDs, entities, variables, numeric policy, RNG), re-exporting submodules with minimal, stable API.

Success: Other crates (vm\_io, vm\_algo, vm\_pipeline, vm\_report, vm\_cli) depend only on vm\_core types/traits—no I/O here; builds on all targets with/without the optional serde feature.

## **2) Scope**

In scope: module declarations, pub use re-exports, core result types, deterministic ordering helpers, numeric/rounding traits, seeded RNG handle, small error enums (core-only).

Out of scope: file/JSON I/O, CLI, state machine orchestration, report formatting.

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

Inputs: none at runtime (library crate).

Outputs: public API:  
 ids::{RegId, UnitId, OptionId, TallyId, ParamSetId, ResultId, RunId, FrontierId},  
 entities::{DivisionRegistry, Unit, Option, …},  
 variables::{VmVar, Params},  
 determinism::{StableOrd, HashCanon},  
 rounding::{Ratio, compare\_ratio\_half\_even},  
 rng::{TieRng}.

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

*(Core provides types; vm\_io owns serialization.)*

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

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

IDs & parsing:

pub fn parse\_reg\_id(s:&str) -> Option<RegId>

pub fn parse\_unit\_id(s:&str) -> Option<UnitId>

pub fn parse\_option\_id(s:&str) -> Option<OptionId>

Deterministic ordering helpers:

pub fn cmp\_options(a:&Option, b:&Option) -> Ordering // by order\_index then id

pub fn sort\_units\_stable(ids:&mut [UnitId])

Numeric policy:

pub struct Ratio { pub num:i128, pub den:i128 }

pub fn compare\_ratio\_half\_even(a:&Ratio, b:&Ratio) -> Ordering

RNG (ties only):

pub struct TieRng(ChaCha20Rng);

pub fn tie\_rng\_from\_seed(hex64:&str) -> Result<TieRng, CoreError>

impl TieRng { pub fn choose<T:StableOrd>(&mut self, slice:&[T]) -> usize }

Variables:

pub struct Params { /\* VM-VAR map materialized into typed fields \*/ }

pub fn params\_default() -> Params

pub fn validate\_params(p:&Params) -> Result<(), CoreError> // domain checks only

Hash canon (interface only; implementation in downstream if needed):

pub trait HashCanon { fn canonical\_bytes(&self) -> Vec<u8>; }

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

pub mod ids; — newtypes + parsers + regex guards for REG:/U:/OPT:/TLY:/PS:/RES:/RUN:/FR:.

pub mod entities; — structs for core entities; Option includes order\_index:int.

pub mod variables; — Params + VmVar enums/constants; domain-level validators (no cross-artifact checks).

pub mod determinism; — StableOrd trait; comparators for units/options; canonical sorting utilities.

pub mod rounding; — Ratio, compare\_ratio\_half\_even, integer/rational comparison helpers.

pub mod rng; — TieRng wrapper over ChaCha20 seeded from hex64; no OS entropy.

pub use re-exports from these modules for downstream crates.

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

Downstream crates import vm\_core::\* for types/traits → vm\_io handles I/O & schema validation → vm\_algo uses rounding, rng, StableOrd → vm\_pipeline orchestrates.

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

**Stable total orders**: Units by UnitId; Options by order\_index then OptionId.

**No float comparisons**: expose ratio/int APIs; round **half-to-even** only at defined comparison points.

**RNG**: only via TieRng with explicit 64-hex seed; no time/OS RNG.

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

ID parsers strictly validate prefix and shape; return None/Err on mismatch.

Ratio constructors must reject den ≤ 0 and normalize sign (store positive den).

tie\_rng\_from\_seed rejects non-hex/invalid length.

cmp\_options must be total and stable even with equal order\_index (break ties by OptionId).

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

ID parsing round-trips: valid shapes parse; malformed shapes fail.

Option ordering: (order\_index, id) sorting stable; deterministic across platforms.

Ratio comparisons: property tests confirm transitivity; tie cases follow half-even rule.

RNG: same seed → identical choice sequences; different seeds → different sequences; no panics on empty slices (return error).

Params default values match Doc 2 defaults; domain checks reject out-of-range percentages or inconsistent combos (random tie without seed not checked here—pipeline enforces when chosen).