# **Pre-Coding Essentials (Component: crates/vm\_core/src/ids.rs, Version/FormulaID: VM-ENGINE v0) — 23/89**

## **1) Goal & Success**

Goal: Provide **typed, validated, and comparable** ID newtypes for all canonical entities, with zero ambiguity and stable ordering.

Success: Every ID parses/prints round-trip, enforces allowed charset/shape, exposes helpers (e.g., UnitId::reg\_id(), UnitId::parent()), and offers Ord/Hash/FromStr/Display. No I/O. Optional serde support behind feature.

## **2) Scope**

In scope: ID types, regex/validators, constructors, Display/FromStr/TryFrom, stable ordering, light helpers (split/parent/join), size guards.

Out of scope: file system paths, JSON I/O (lives in vm\_io), heavy normalization beyond spec shapes.

## **3) Inputs → Outputs**

Inputs: ASCII strings from loaders/tests.

Outputs: Strong types used across core/algo/pipeline: RegId, UnitId, OptionId, TallyId, ParamSetId, ResultId, RunId, FrontierId, AutoPkgId.

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

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

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

**Parsing/constructors**

impl FromStr for RegId/UnitId/OptionId/...

pub fn RegId::new(name:&str, version:&str) -> Result<Self, IdError>

pub fn OptionId::new(slug:&str) -> Result<Self, IdError>

pub fn TallyId::new(name:&str, ver:u32) -> Result<Self, IdError>

pub fn ParamSetId::new(name:&str, semver:&str) -> Result<Self, IdError>

pub fn ResultId::from\_hash(short:&str) -> Result<Self, IdError>

pub fn RunId::new(ts\_utc:&str, short:&str) -> Result<Self, IdError>

pub fn FrontierId::from\_hash(short:&str) -> Result<Self, IdError>

pub fn AutoPkgId::new(name:&str, ver:u32) -> Result<Self, IdError>

**Unit helpers**

pub fn UnitId::reg\_id(&self) -> &RegId

pub fn UnitId::path(&self) -> &[String]

pub fn UnitId::is\_root(&self) -> bool

pub fn UnitId::parent(&self) -> Option<UnitId>

pub fn UnitId::with\_child<S:AsRef<str>>(&self, seg:S) -> Result<UnitId,IdError>

**Common**

pub fn is\_valid\_short\_hash(s:&str) -> bool

pub fn is\_valid\_semver\_in\_id(s:&str) -> bool // PS/AP

pub fn enforce\_ascii\_and\_len(s:&str) -> Result<(),IdError>

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

**Newtypes**: tuple structs over SmolStr/String (SmolStr if you want small-string optimization; otherwise String).

**Regexes** (compiled once w/ lazy\_static/once\_cell):

REG: ^REG:[A-Za-z0-9.\_-]+:[A-Za-z0-9.\_-]+$

OPT: ^OPT:[A-Za-z0-9.\_-]+$

TLY: ^TLY:[A-Za-z0-9.\_:-]+:v[0-9]+$

PS: ^PS:[A-Za-z0-9.\_-]+:v(0|[1-9]\d\*)\.(0|[1-9]\d\*)\.(0|[1-9]\d\*)(?:[-+][A-Za-z0-9.-]+)?$

RES: ^RES:[A-Za-z0-9.\_-]+$

RUN: ^RUN:\d{4}-\d{2}-\d{2}T\d{2}-\d{2}-\d{2}Z-[A-Za-z0-9.\_-]+$ *(timestamp uses - in place of :)*

FR: ^FR:[A-Za-z0-9.\_-]+$

AP: ^AP:[A-Za-z0-9.\_-]+:v[0-9]+$

**Unit**: Two-stage check: surface ^U:REG:[A-Za-z0-9.\_-]+:[A-Za-z0-9.:\_-]+$, then **structural** validation that the embedded REG: equals the RegId part and that each path segment is non-empty.

**FromStr/Display**: strict parse; Display prints canonical form, preserving case.

**Ordering/Hashing**: derive Eq, Ord, Hash; ordering is lexicographic on full string, which matches spec’s stable orders.

**Helpers**: UnitId::parent() finds last : after the embedded REG:…; with\_child appends :<seg> after validating seg.

**Serde** (behind feature = "serde"): #[serde(transparent)] + visitor parsing via FromStr.

**Guards**: ascii\_only, max\_len, and **no** NULs; return IdError::{TooLong, NonAscii, BadShape, EmptySegment, MismatchedRegistry, BadSemver}.

## **8) State Flow**

Loaders read raw strings → call str::parse::<…Id>() → downstream code uses typed IDs for maps/sorts/joins with no re-validation.

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

**Determinism**: IDs are **case-sensitive** ASCII and totally ordered lexicographically; Option ordering elsewhere breaks ties by OptionId after order\_index.

**No numeric math** here.

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

Empty or over-length strings ⇒ TooLong/BadShape.

Non-ASCII (e.g., whitespace or Unicode) ⇒ NonAscii.

**UnitId** whose embedded REG: does not match supplied RegId (when constructing from parts) ⇒ MismatchedRegistry.

Path with empty segment ("U:REG:X::Y") ⇒ EmptySegment.

RunId timestamp not YYYY-MM-DDT HH-MM-SSZ (with dashes instead of colons) ⇒ BadShape.

ParamSetId semver fails regex ⇒ BadSemver.

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

Round-trip: for each ID kind, format!("{}", s.parse::<Id>()?) == s.

Negative cases per regex (lowercase reg:; non-ASCII; spaces; empty segments) fail with correct IdError.

UnitId::parent():

root returns None; two-level returns correct parent; multi-level works.

UnitId::with\_child():

rejects empty or invalid child; preserves RegId.

Ord stability:

Sorting Vec<UnitId> is stable and matches raw lexicographic order.

Sorting Vec<OptionId> matches lexicographic.

Serde (if enabled): serialize → string; deserialize → identical ID.

DoS guard: strings of length > ids.max\_len rejected fast.