Annex A — Variable Canonical Reference Table.md

File

Annex B — Canonical Test Pack.md

File

Annex C — Glossary & Definitions.md

File

Doc 1 — Database Specification (Entities, Fields, Relationships).md

File

Doc 2 — Common Variables Specification (Core, Operational Defaults, Advanced Controls).md

File

Doc 3 — Technical Platform & Release Policy.md

File

Doc 4 — Algorithm Specification (Steps, Allocation, Gates & Edge Cases).md

File

Doc 5 — Processing Pipeline Specification (State Machine & Functions).md

File

Doc 6 — Test Specifications (Allocation, Gates, Frontier & Determinism).md

File

Doc 7 — Reporting Specification (Structure, Templates & Visual Rules).md

File

here is the doc for an app. This is the fixed reference. Later on I will ask you to validate code alignment. prepare yourself

Thought for 1m 7s

**ChatGPT said:**

Got it. I’ve loaded your fixed reference and I’m ready to validate code against it when you say go. Below is the concise alignment rubric I’ll apply, mapped to the docs you provided.

**Alignment rubric (what I’ll check)**

1. Canonical JSON & hashing (engine + artifacts)

* Artifacts must be canonical JSON (UTF-8, LF, **sorted keys**, arrays per Doc 1A ordering). IDs/hashes must match the canonical payloads.

1. FID / Normative Manifest lock

* FID is computed from outcome-affecting rules + **Included** VM-VARs only; **Excluded** vars never change FID. Manifest must be canonicalized, sorted by numeric VM-VAR ID.
* Recomputed FID must equal both Result.formula\_id and RunRecord.formula\_id.

1. Variable sets (Included vs Excluded)

* **Included (in FID)** membership per 2A/Annex A (e.g., 001–007, 010–017, 020–031 incl. 021/029–031, 040–049, **050**, 073).
* **Excluded (non-FID)** presentation/report toggles and seed: 032–035, **052**, 060–062.

1. Pipeline state machine & ordering (no nondeterminism)

* Execution order S0→S7 exactly as in Doc 5A; loops honor canonical ordering; self-verify before exit.

1. Gates → Frontier → Allocation → Ties (flow discipline)

* Per-unit flow: GATES (4B) → FRONTIER (4C) → ALLOCATE (4A) → if tie then TIES (4C). Labels after allocation (presentation only).

1. Ties & RNG (determinism)

* tie\_policy **050** is in FID; tie\_seed **052** is Excluded (recorded only if random tie occurs). RNG draws happen **only** when 050="random" and an actual tie exists; echo rng\_seed iff any random tie.
* Changing **050** changes FID; changing **052** alone does **not**.

1. Test harness conformance (Doc 6)

* CLI invocation interface; required outputs; canonical checks; FID/ID checks; acceptance template; suite behaviors (6A/6B/6C).
* Specific validation/error cases (e.g., order\_index uniqueness, vote sums, FID lock).

1. Reporting/Renderer rules (Doc 7A)

* Renderer reads canonical artifacts **read-only**, never recomputes allocations; obeys toggles 032–035, 060–062; section ordering affects report only, not JSON.
* Required disclosures (FID, Engine Version, Variant) and conditional sections (Ties/Frontier/Sensitivity).

1. CLI & exit codes (Doc 3A/5A)

* Contract: vm\_cli --registry ... --tally ... --params ... --out ... (optional --seed override). Exit codes 0/2/3/4/5 mapped per spec.

1. Field catalog & integrity (Doc 1B)

* Referential integrity across artifacts; deterministic order\_index; ID formats and digests; UTC timestamps.

**What I’ll need from the code side (when you ask to validate)**

* A runnable CLI (or entrypoint) matching Doc 6A invocation and Doc 3A contract.
* Produced artifacts (result.json, run\_record.json, optional frontier\_map.json) for at least the canonical test cases from Annex B.
* The ParameterSet used, to recompute FID from the **Included** set (Annex A).

When you’re ready, share the build/run outputs (or point me at the repo/paths), and I’ll run this checklist end-to-end against your engine and the renderer.

Sources

**You said:**

data\_tests.txt

Document

is this well aligned?

Thought for 1m 29s

**ChatGPT said:**

Short answer: mostly aligned, with a few gaps and a couple of naming/layout nits.

**What’s aligned**

* **Gates run before allocation; failures mark the unit Invalid and skip allocation/frontier.** Your vm\_tst\_005\_pipeline\_order\_and\_stop\_rules intention matches the canonical stage order and branching rule.
* **Gate-order semantics & reason-recording.** The skeleton frames sanity → eligibility → validity, which the spec fixes (and requires ordered reason recording).
* **Use of Annex-B fixtures.** You point tests at “annex\_b … gates …” fixtures, which is consistent in spirit with Annex-B being the oracle for 6A–6C.
* **Allocation/method checks.** The WTA and LR/D’Hondt/Hare convergence intentions belong in the 6A allocation-correctness family.
* **One-decimal diagnostic formatting.** Your helper renders a single decimal for percentages; Doc-7 uses pct1 for shares in reports (presentation). Using one-decimal for test diagnostics is fine.

**Gaps / risks to fix**

1. **Fixture paths and IDs.** Annex-B fixes the repo layout as /annex-b/cases/<CASE-ID>/{registry,tally,params}.json. Your constants point to fixtures/annex\_b/gates/s00x/.... Consider aligning paths and adopting **VM-TST-2xx** IDs for gates/frontier cases so a runner can rely on Annex-B’s manifest without remapping.
2. **Approval-rate denominator anchor.** The test comment asserts a **fixed** denominator “approvals\_for\_change / valid\_ballots” for majority. I couldn’t find that exact normative sentence in the provided excerpts; gates/thresholds are specified, but the denominator wording isn’t shown verbatim. If this rule is intended to be normative, cite its clause in **Doc-4B** or **Annex-A**; otherwise, add that sentence to the spec (or relax the test to whatever the spec actually states).
3. **Harness-level assertions from Doc-6A §2.3 are missing here.** You’ll also need tests that verify: canonicalization + IDs (RES:/RUN: rules), **FID recomputation** vs Result.formula\_id/RunRecord.formula\_id, **vars\_effective** echo, and “no RNG” for 6A. (Maybe they exist elsewhere; they aren’t in this file.)
4. **Gate failures vs schema errors.** Add explicit assertions that **gate failures** keep process exit 0 with Result and RunRecord present, while **schema/ref/order violations** cause exit 2 with specific codes (e.g., E-BT-SUM, E-DR-ORD-UNIQ). Your comment hints at this—make it an explicit check.
5. **Reason ordering determinism.** Add a check that failing reasons are recorded in ascending **VM-VAR** order, then symbolic tokens, to guarantee byte-identical RunRecord.summary.
6. **Frontier pre-check behavior.** Where Frontier is enabled, include a case that exercises frontier\_missing\_inputs as a **validity failure** and confirms FrontierMap is emitted **only** when 040 != "none" and 034=true.
7. **Ordering invariants.** For allocation tests, assert units sorted by unit\_id and allocations by order\_index to pin determinism.

**Minor nits**

* The TODO in vm\_tst\_005 mentions “skip TABULATE..FRONTIER”. Spec terms the stages **S2 gates → S3 frontier → S4 allocation**; consider using spec labels in messages.
* If you keep diagnostics that print percent values, note they’re **non-FID** (presentation). Keep structural assertions on the underlying numbers/flags.

**Bottom line**

* Test **intent** ≈ spec intent. Align fixture **layout/IDs**, add harness-level checks from Doc-6A, and either cite or relax the **approval-rate denominator** rule. With those, this will be well aligned to Docs **4–6** and Annex **A/B**.

Sources

**You said:**

so, which files needs fix?

**ChatGPT said:**

From the check I just ran on your data\_tests.txt against the fixed reference docs, the pieces that need fixing or supplementation are:

**Files / areas needing fixes or additions**

1. **Annex B fixture references**
   * Your current test code points to fixtures/annex\_b/gates/... with ad-hoc names.
   * Needs to match Annex B’s fixed repo layout /annex-b/cases/<CASE-ID>/{registry,tally,params}.json and use canonical VM-TST IDs.
2. **Approval-rate denominator rule**
   * The comment in your gate test asserts a fixed denominator (“approvals\_for\_change / valid\_ballots”) for majority.
   * I couldn’t find this sentence verbatim in the normative docs you gave. Either:
     + cite the correct clause in Doc 4B or Annex A, or
     + remove/relax that assumption in the test.
3. **Doc 6A harness-level checks**
   * This file lacks tests for:
     + canonicalization of artifacts
     + RES:/RUN: ID format and hashing
     + FID recomputation from Included vars
     + vars\_effective echo
     + “no RNG” case handling
   * These are mandatory under Doc 6A § 2.3 and should be added.
4. **Exit-code semantics**
   * Add explicit tests that:
     + Gate failures → exit 0 with Result + RunRecord present.
     + Schema/ref/order violations → exit 2 with correct error codes.
5. **Reason ordering determinism**
   * Add a check that failing reasons in RunRecord.summary are ordered by numeric VM-VAR ID, then symbolic tokens.
6. **Frontier pre-check cases**
   * Include a case where frontier\_missing\_inputs triggers a validity failure and confirm FrontierMap is only emitted when 040 != "none" and 034 = true.
7. **Allocation ordering invariants**
   * Ensure allocated units are sorted by unit\_id and allocations by order\_index.

If you want, I can now produce a **targeted fix list** mapped to the exact spec section for each point so you can hand it directly to whoever is maintaining the tests. That way you’ll know exactly which code files and which clauses they must match.

2/2

**You said:**

manifest.txt

Document

root.txt

Document

schemas.txt

Document

vm\_algo.txt

Document

vm\_app.txt

Document

vm\_cli.txt

Document

vm\_core.txt

Document

vm\_io.txt

Document

vm\_pipeline.txt

Document

vm\_report.txt

Document

here are the other files. Analyse it all, make a big bug report. At the end, list the files needing reworking.

**ChatGPT said:**

To create a complete and accurate bug report across the codebase, could you please clarify:

1. Which files (if any) should be treated as authoritative test outputs or expected results? Are you using Annex B JSONs for truth?
2. Should I focus solely on engine correctness (FID alignment, canonical output, stage order, determinism), or include UI, CLI usability, or build tooling?
3. Are there any known failing test IDs, flaky behaviors, or mismatches you want prioritized?

Once confirmed, I’ll produce a structured full bug report and remediation plan.

**You said:**

the source of truth is the 10 files (7 doc and 3 annex) be thorough while not looking to optimise code. It's debugging.

**ChatGPT said:**

Understood. I’ll audit the codebase (engine, CLI, pipeline, schemas, report model, and test scaffolds) for strict conformance against the 10 source-of-truth files: Docs 1–7 and Annexes A–C.

I'll focus entirely on correctness and fidelity to spec — no performance tuning or stylistic optimizations — and produce a comprehensive bug report. This will include all observed mismatches, skipped steps, hash or ordering violations, and any non-conformant behaviors. I’ll also clearly list which files require fixes and what aspects are affected.

I'll begin this analysis now and notify you when the report is ready.

Research completed in 11m · 9 sources

Codebase Debug Audit Against VM Spec and Annexes

**1. CLI & Input Contract (Doc 3A/3B):** The CLI argument handling violates several spec requirements. The --validate\_only flag is parsed but never honored – the code always executes a full run instead of performing a dry input validation as required. Error exit codes are not differentiated per spec: all CLI errors return exit code 1, contrary to Doc 5B’s mandates (e.g. input validation errors should exit 2, hash mismatches exit 3, I/O errors exit 4, etc.). For example, a schema validation failure or missing file yields a generic exit 1 (via ExitCode::from(1)). The CLI also offers --ballots (raw ballots) input, but the engine enforces tally-only mode – any manifest synthesized with ballots\_path is rejected by ensure\_manifest\_contract (“must specify ballot\_tally\_path”). This makes --ballots unusable (both or neither of --ballots/--tally triggers an error). In short, the CLI does not cleanly support raw ballots despite exposing the flag. Additionally, while network paths are correctly blocked (no http:// allowed), the CLI does not propagate the spec’s explicit engine exit codes or fully implement quiet/validation-only modes. Affected files: *vm\_cli/src/args.rs*, *vm\_cli/src/main.rs*. Rework: implement proper --validate\_only short-circuit, map errors to spec exit codes, and remove or fix the unsupported --ballots flag handling.

**2. Normative Manifest & FID Construction (Doc 1A §2.3, Doc 5A, Annex A):** The engine’s Formula ID computation is misaligned with the normative rules. Instead of hashing the **Included VM-VAR key/value map**, the code builds a “Normative Manifest” containing input file digests. This is incorrect – per spec the FID must reflect *all outcome-affecting variables and rules* (001–007,010–017,…,050,073) and exclude data inputs. The current implementation hashes a JSON like {"normative\_inputs": {"division\_registry\_sha256": "...", ...}}, meaning the FID in this implementation wrongly depends on the specific registry/tally contents (violating the formula independence from run data). It entirely omits the VM-VAR entries. In fact, PipelineCtx.nm\_canonical is left as an empty JSON object in run\_from\_manifest\_path, causing compute\_formula\_id to produce a fallback "000...000" FID (64 zeros). This is a clear violation: the code neither serializes included variables in sorted order nor incorporates default values as required. Also, excluded presentation toggles (032–035,060–062) are trivially omitted (since *no* variables are hashed at all), but the **inclusion logic is effectively stubbed out**. The manifest hashing routine even wraps errors by returning 64 zeros instead of failing, which masks problems instead of enforcing correctness. In summary, the engine is not producing a legitimate Formula ID. Affected files: *vm\_pipeline/src/lib.rs* (compute\_formula\_id, run\_from\_manifest\_path), *vm\_pipeline/src/load.rs* (compute\_nm\_fid\_if\_present\_from\_digests). Needs fixing to serialize all Included VM-VARs (001–007,…,073 except excluded) in numeric order with their values (defaults if not set) before hashing, and to remove input-specific hashes from FID computation.

**3. Parameter Set Defaults & Variable Inclusion (Annex A, Doc 2A/2B):** The handling of VM-VAR inputs and defaults is incomplete. Spec requires that all **Included** vars have effective values (using canonical defaults if absent), and that the ParameterSet covers them (conformance C-2A-INC). The code does enforce presence of all included fields via the Params struct (all included variables are non-Option fields), but there is no logic to inject defaults if a field is missing – parse would just fail. In other words, the engine doesn’t gracefully apply default values; it assumes the input JSON always provides every required var. This rigidness meets C-2A-INC but technically conflicts with the spec’s expectation that defaults be applied if some included vars are omitted. Additionally, domain enforcement is only partially implemented. For example, VM-VAR-003 (precision 0–6) is stored as u8 but nothing in code prevents an out-of-range value. Likewise, percentage fields (Pct type) and Enums (like tie\_policy) rely on serde/JSON Schema validation rather than explicit checks. There is no explicit range check for Pct or numeric domains in the Rust code – any invalid value that passes JSON schema could propagate. Moreover, presentation toggles (032–035, 060–062) are effectively ignored in engine execution. VM-VAR-034 (frontier\_map\_enabled) and 035 (scenario comparison) are Excluded from FID but should still alter behavior (034 triggers FrontierMap output; 035 triggers scenario analysis per Doc 5B). The current engine never reads or acts on these toggles: e.g. setting 034=true in input does not result in any frontier map artifact (the code leaves frontier\_map: None unconditionally), and 035=true (scenario analysis) is not implemented at all. In short, the engine doesn’t honor the operational/presentation VM-VARs that should toggle optional outputs or analyses. Affected files: *vm\_core/src/variables.rs*, *vm\_io/src/loader.rs* (ParameterSet parsing), *vm\_pipeline/src/lib.rs* (use of 034/035).

**4. Pipeline Stage Flow & Order (Doc 5A/5B):** The high-level pipeline is mostly scaffolded and violates the strict stage sequencing and completeness defined in Doc 5. Key stages are missing or placeholder: there is no real **Tabulate or Allocate stage** – the code never calls any function to count votes or assign seats (no invocation of vm\_algo’s tabulation/allocation in the pipeline). After loading inputs and computing FID, the code jumps directly to labeling and output assembly. This skips Doc 5A steps 4–8 entirely. The Stage 3 “Count/allocate” is absent, meaning any tally data is unused and the Result is not actually populated with per-unit outcomes. Likewise, **ApplyGates (eligibility/validity)** and **FrontierHook** are not invoked (no call to check thresholds or to map\_frontier unless one manually integrates it). The LabelDecisiveness step is hard-coded: the pipeline just marks every run as “Decisive” by default, ignoring the Doc 5C criteria for Marginal/Invalid outcomes. For example, an election that should be labeled *Invalid* due to gate failures would still get “Decisive” since no gate logic runs and LabelBlock.reason is always None. The flow discipline is also broken for “manifest vs explicit” inputs: the pipeline’s load\_normative\_from\_manifest path computes nm\_digest and formula\_id, but the explicit file path path (load\_normative\_from\_paths) sets both to None (omitted). In practice the CLI never calls load\_normative\_from\_paths (it synthesizes a manifest even for explicit mode), but this means if the pipeline API were used directly with files, it would produce outputs missing the required formula\_id. This asymmetry violates the spec’s rule that **every** run’s outputs include the FID. Finally, the pipeline never performs the final self-verification step (VM-FUN-014): there’s no check that re-hashing the output yields the same IDs or recomputing FID from nm\_digest matches the recorded FID. The spec requires an internal consistency check (and exit 3 on mismatch), which is not present. Affected: *vm\_pipeline/src/lib.rs*, *vm\_pipeline/src/tabulate.rs*, *vm\_pipeline/src/allocate.rs* (not utilized). Full conformance demands implementing all missing stages in order (Tabulate → ApplyGates → Frontier → Allocate → ResolveTies → Label) and performing the final verification before exiting.

**5. Allocation Algorithm, Gates, Frontier, Ties (Docs 4A–4C, 6A–6C):** The core election logic is unimplemented or incorrect, leading to multiple spec violations:

* **Seat Allocation & Share Calculation:** The engine defines data structures for various PR methods (e.g. Dhondt, Sainte-Laguë in *vm\_algo/src/allocation/*.rs) but never uses them in the pipeline. There is no enforcement of algorithm\_family (VM-VAR-001) – e.g. a mixed-member proportional run vs. plurality run are treated no differently. Rounding policy (002) and precision (003) are similarly ignored; any fractional seat shares or vote ratios are not computed at all. This contradicts Doc 4A which fixes how these variables guide allocation rounding and seat distribution.
* **Preconditions and Gate Enforcement:** Validity gates (010–017, 020–029) and run scope (021) are not applied. For instance, units that should be invalidated for failing quorum or majority thresholds remain “Decisive” with allocations presumably empty (the engine would output no allocations simply because it never computed any). The code includes a validate() scaffold in *vm\_pipeline/src/validate.rs* that checks option ordering and some structure, but all substantive gate checks (e.g. turnout below 50% → invalidate unit, etc.) are commented out or TODOs. Protected area overrides (045) and autonomy (046) are defined in the Params but the logic to apply them (Doc 4C) is absent. The code doesn’t mark protected units or enforce that 045=“allow” bypasses only eligibility gates (C-6B-PROT). It also doesn’t record protected\_bypass reasons anywhere.
* **Frontier Contiguity Analysis:** The engine provides a map\_frontier function (in *vm\_pipeline/src/map\_frontier.rs*) that implements band assignment, mediation, enclave detection, etc., but this is never wired into the run. VM-VAR-040 (“frontier\_mode”) is effectively ignored – the pipeline doesn’t call map\_frontier even if 040≠“none”. Additionally, required behavior when frontier inputs are missing is not handled. Spec mandates that if a frontier mode that needs adjacency data is enabled but no adjacency is provided, the engine must not crash or silently miscompute – it should flag an error reason "frontier\_missing\_inputs" in the output (per Doc 6B and C-6B-FRONTIER). The current code would either just use an empty adjacency (since loader makes adjacency optional) or, more likely, never notice because frontier isn’t executed. In either case, it fails to output the required diagnostic. Moreover, when frontier is executed, the engine should emit a FrontierMap artifact only if 034=true and frontier actually ran. In this implementation, even if 034=true, frontier\_map.json is never produced due to the stubbed logic. This breaks Doc 5B (VM-FUN-012) and conformance tests expecting a frontier\_map in “fat” mode when enabled. Affected areas: *vm\_pipeline/src/map\_frontier.rs*, *vm\_pipeline/src/lib.rs* (frontier\_map: None placeholder), *vm\_algo* gating logic.
* **Tie Resolution & Deterministic RNG:** The handling of ties (equal scores for final seats) does not meet Doc 4C/5C. VM-VAR-050 (tie\_policy) is included in FID, but the code never actually branches on it – there is no implementation of status-quo vs deterministic vs random tie-breaking. The vm\_core::rng module defines a TieRng and Annex B pinning (likely using rand\_chacha), but the pipeline doesn’t invoke it. As a result, ties would either remain unresolved or be implicitly decided by map iteration order. Specifically, for tie\_policy="random", spec requires drawing exactly *k* 64-bit numbers for a k-way tie and sorting by (draw, option\_id), recording the seed in RunRecord.determinism.rng\_seed if used. None of this happens – the RunRecord.policy only echoes the seed value that was input (v052), but since no randomization occurs, it’s meaningless. The code also fails to log tie events. Per Doc 5C/6C, every occurrence of a tie that had to be broken should add an entry to RunRecord.ties[] with the unit\_id, tie policy, and (if random) the seed used. The RunRecordDoc struct has no ties field at all. This is a direct omission of a required audit trail. In effect, tie determinism and reproducibility can’t be verified because the engine neither resolves ties according to spec nor documents any tie-breaking decisions. Affected files: *vm\_algo/src/resolve\_ties.rs* (likely empty), *vm\_pipeline/src/lib.rs*, *vm\_pipeline/src/build\_run\_record.rs*. These need full implementation of tie-breaking per 050/052 and logging each tie per unit (including rng\_seed only when policy=random, see Annex A §4).

**6. Output Artifacts & Schema Mismatches (Doc 1A, Annex C):** The JSON outputs produced by the engine do not conform to the documented schema and content requirements:

* **Result.json Content:** The ResultDoc the code writes is extremely minimal – it contains only formula\_id and a top-level decisiveness label. It lacks the per-unit results and summary metrics mandated by the schema. According to *result.schema.json*, result.json must have schema\_version, an engine version, a creation timestamp, a summary of totals, and a units array of results (with each unit’s allocations and its own “Decisive/Marginal/Invalid” label). The current code populates none of these. No schema\_version field is added, engine\_version is missing (it should mirror the EngineMeta, e.g. “v0.1.0”), and created\_at is not recorded (the RunRecord has timestamp\_utc, but the Result should also have one per schema). Crucially, the engine never builds the units list of allocations in the Result – it doesn’t even have a data model for unit results in ResultDoc as implemented. Thus, the output result.json violates the required structure and omits essential outcome data. It also means the result\_id is computed over an incomplete payload. Files: *vm\_pipeline/src/lib.rs* (Result assembly).
* **RunRecord.json Content:** The RunRecordDoc is missing fields and has a different shape than specified. The schema (Doc 1B/Annex C) expects a RunRecord with sections for inputs (including digests and IDs), vars\_effective (a map of all included VM-VARs and their effective values), policy (tie-break policy and seed), and ties (the list of tie events), among others. In the code, RunRecordDoc has no vars\_effective at all – it only stores formula\_id and normative\_manifest\_sha256 as separate fields. This fails conformance **A-REG-FID** which requires that recomputing FID from the included vars map equals the stored formula\_id. Since the run record doesn’t actually list the var values used, an external verifier cannot recompute the FID from it (the nm\_digest alone is insufficient without the manifest content). Similarly, the ties array is missing (as noted, no ties field), violating the requirement to log random tie outcomes (Doc 5C and **A-REG-TIES**). Additionally, the engine subsection in RunRecord should include at least the engine version and the FID or manifest hash used (often called formula\_manifest\_sha256 in schema). The code instead puts formula\_id and normative\_manifest\_sha256 at top-level fields of RunRecordDoc, not under an engine or platform section. The schema’s required platform info (OS, architecture, etc.) is entirely absent – the implementation doesn’t capture runtime environment at all. Moreover, the IDs for inputs are placeholders (“REG:local”, “TLY:local”, etc.), rather than actual content-based IDs. Spec says these should be the canonical hashes or assigned IDs of the input files. The code acknowledges these are placeholders in comments. Using non-canonical dummy IDs violates the cross-referencing rules (e.g. DivisionRegistry.id should match the RunRecord’s inputs.division\_registry\_id). Affected: *vm\_pipeline/src/lib.rs* (RunRecord assembly). This needs to include a full vars\_effective map (with all Included var IDs and their values), populate the ties list, nest the engine metadata as per schema, and use real input IDs (e.g. REG:<sha256> of registry.json, which vm\_io can provide).
* **ID Formatting:** The code computes result\_id and run\_id correctly in format (RES:<64hex> and RUN:YYYY-MM-DDTHH:MM:SSZ-<hex>), using canonical bytes. However, it never computes a frontier\_id when a FrontierMap is produced, and it doesn’t produce a manifest ID (MAN:<hash>). Annex B’s test oracle uses a manifest hash (sometimes called formula\_manifest\_sha256 or MAN:) to verify FID recomputation. In the current RunRecord, normative\_manifest\_sha256 is present (this is essentially nm\_digest), but it’s not labeled as a MAN:ID nor exposed for external recomputation except as a raw hash string. It should be part of the engine metadata and follow the MAN:<hex> format per schema. Also, because the Normative Manifest itself was wrong (point #2), the nm\_digest here doesn’t correspond to an Annex A manifest of var values – it’s a hash of input file hashes. So both the content and labeling of this field are off-spec. In summary, all output artifacts need extensive fixes to match the canonical schemas in Annex C. Files: *schemas/*.json (for reference)\*, *vm\_pipeline/src/build\_result.rs*, *vm\_pipeline/src/build\_run\_record.rs*.

**7. Report Renderer & Presentation (Doc 7A/7B):** The post-processing report (JSON/HTML) does not fully meet the requirements for disclosure and toggle handling. While the vm\_report module defines a structure with all required sections (Cover, Snapshot, Eligibility, BallotMethod, Legitimacy, Outcome, Frontier, Sensitivity, Integrity), the data filling those sections is rudimentary or nonexistent:

* **Use of Toggle Variables (060–062):** These VM-VARs control report labeling and language but are not threaded through. For instance, **VM-VAR-060** (likely the decisiveness margin in tenths) and **061** (margin mode, e.g. “dynamic\_margin”) are intended to influence whether an outcome is marked Marginal vs Decisive in the report. The engine, however, never computes any margin or uncertainty – it unconditionally labeled the result Decisive. Thus, the renderer has no input to decide differently, effectively ignoring 060/061. Similarly, **VM-VAR-062** (display language or bilingual toggle) is not connected; the HTML renderer always sets bilingual: None and embed\_assets: true by default, with no option to choose language. If 062 were meant to select a language code or “auto”, the code does not use it to pick a language or load translations. This is a violation of Doc 7’s internationalization and labeling rules – for example, if 062=”fr” was set, the report should use French labels, etc., which is not implemented.
* **Required Content in Narrative:** Doc 7 requires certain explanations in the report. The code comments note one (for approval ballots, include a sentence about “approvals / valid ballots”), suggesting an attempt to comply. But other required narratives or disclosures may be missing. For example, the spec likely mandates that the Formula ID or engine version be disclosed in the report for transparency. The current JSON report model doesn’t include the formula\_id or engine hash at all (ReportModel has no field for FID or manifest hash), meaning the public report does not inform readers of the formula used – a potential compliance gap with Doc 7’s transparency guidelines. Also, the *Integrity* section should list any integrity checks or data sources (Doc 7C); the model has an SectionIntegrity but the code to populate it is minimal, likely just echoing input hash totals without commentary.
* **Scenario/Sensitivity Analysis:** The renderer model has an optional SectionSensitivity for scenario comparisons (what-if outcomes). This corresponds to VM-VAR-035 usage. However, since the engine never runs CompareScenarios (and in fact doesn’t implement it), the report will never include a sensitivity analysis even if 035=true. This omission means any “appendix” data required by Doc 5B VM-FUN-013 and Doc 7 (perhaps an alternate outcome if one variable changed) is not shown, even though the structure is prepared for it. In tests, a case where 035=true would expect a “Scenario” or “Sensitivity” section; the engine would fail to produce one.
* **Ordering and Formatting:** The JSON and HTML output ordering seems to follow spec (sections are in a fixed order in ReportModel), and keys are sorted via canonical JSON on output. However, the **RunRecord/Result linking** in the report might not meet disclosure rules. Doc 7 might require that any deviations or non-deterministic elements (like use of random seed) be explicitly mentioned in the report narrative. The code does not add any text about randomness if a tie was resolved by RNG, nor does it mention if any gates caused exclusions – because that info wasn’t forwarded from the engine. Essentially, the renderer cannot fulfill Doc 7’s “explain all outcomes” mandate because the engine isn’t providing the necessary context (e.g., which units were invalidated and why, what tie-break method was used).

In summary, the Report output is not fully compliant: it ignores the reporting toggles, omits required fields (FID, engine version) and sections (scenario analysis), and cannot convey certain decisions (random tie-breaks, gate failures) because the engine never communicates them. Affected files: *vm\_report/src/lib.rs*, *vm\_report/src/render\_json.rs*, *vm\_report/src/render\_html.rs*. These need enhancements once the pipeline supplies complete data.

Each of the above issues points to specific files that require rework, notably **vm\_cli** (args/main), **vm\_pipeline** (core pipeline logic, load & output assembly), **vm\_core/vm\_algo** (algorithm implementation for allocation, gates, ties), **vm\_io** (manifest/FID handling, schema enforcement), and **vm\_report** (render integration of toggles and complete data). The codebase in its current form is not strictly aligned with Docs 1–7 and Annexes A–C; it omits critical spec-mandated behaviors and outputs at every layer of the system.

Sources