Reproducibility Controls

Purpose

Reproducibility in GROW is the operational guarantee that, given a record_id from s3-provenance-metadata-schema, a builder or evaluator can re-derive the same decision path — not necessarily the same token-for-token text — and confirm the output was a faithful function of pinned inputs, pinned code, and pinned models. Without this, regressions are invisible, audits are theatrical, and HITL overrides have nothing to compare against.

This spec defines the version pinning regime, the snapshot regime, the determinism rules, and the re-run runbook.

Version Pinning Regime

Every system that emits provenance records MUST pin, by content hash or immutable tag, all of the following before each run:

Pin	Where stored	Granularity
Source snapshot ref	transformation_history[0].data_shape + S3/blob URI	Per source_id
Lineage map version	version.lineage_map_version	Semver
Code version	version.system	Git SHA or semver tag
Model id + revision	version.model	Vendor-published immutable ref
Prompt template	version.prompt_hash	sha256 over rendered template
Retriever config	retrieval_sources[].retriever_id + hashed config	sha256
Tool registry	per tool_calls[].tool_id	Immutable tool version

Mutable references (e.g., "latest", floating branch tags) are prohibited in production paths. CI rejects systems whose provenance record contains any unpinned ref.

Snapshotting Regime

Sources fall into three snapshot classes:

1. Append-only sources (e.g., ownerrez-bookings, ecoguardian-stream): pin by as_of timestamp + last-record hash. Re-runs read the same window.

2. Mutable sources (e.g., wcpa-parcels, fl-dep-lpa0381): full snapshot at run time, stored content-addressed for the retention window declared in s3-governance-retention-policy.

3. Generated sources (embeddings, summaries upstream of this system): treated as their own Source rows with their own snapshots; never re-generated implicitly.

Snapshots are addressed by sha256(content) and stored under s3://prov/<system>/<record_id>/sources/<source_id>/. Garbage collection follows retention class, not LRU.

Determinism Rules

GROW separates the lineage into deterministic and non-deterministic segments. Both can be reproduced; the rules differ.

Deterministic segments (Extraction, Transformation, ToolCall with deterministic tools): MUST produce byte-identical outputs across re-runs against the same snapshots. Any drift is a bug, not a model issue.

Non-deterministic segments (Embedding with non-deterministic backends, Retrieval with stochastic top-k, Inference): MUST be reproducible at the decision-path level:

- Same decision_origin value for the same inputs at temperature ≤ 0.2 in ≥ 95% of re-runs.
- Same set of retrieval_sources[].result_doc_ids (set equality, not order) for the same retriever config in ≥ 90% of re-runs.
- Same tool_calls[].tool_id sequence in ≥ 95% of re-runs.

These thresholds are the inputs to s2-regression-discipline. Falling below them is a regression event, even if no individual output is wrong.

Where Determinism Is Impossible

If a segment cannot meet the decision-path thresholds (e.g., creative generation, exploratory summarization), it MUST be:

1. Labeled is_deterministic: false in the lineage map.

2. Bracketed by deterministic checkpoints — the inputs to the segment and the outputs after the next deterministic step are both content-hashed.

3. Sampled — at least every Nth run captured into a regression harness for human comparison.

Re-Run Runbook Template

Use this runbook verbatim when reproducing a provenance record. It is the canonical procedure referenced by audits and incident reviews.

```
RE-RUN RUNBOOK — Provenance record <record_id>

PRECONDITIONS
[ ] Record retrieved from provenance store and validated against s3-provenance-metadata-
schema.
[ ] Reviewer has read access to all sources listed in record.source_id.
[ ] Retention window for snapshots has not elapsed (see s3-governance-retention-policy).

STEP 1 — Resolve pins
  1.1 Read record.version.{system, model, prompt_hash, lineage_map_version}.
  1.2 Check out code at version.system. Verify git SHA matches.
  1.3 Resolve model id + revision; confirm vendor revision is still available.
       If revision is unavailable, STOP and file under "vendor pin loss" incident.
```

```
STEP 2 — Rehydrate sources
  2.1 For each source_id, fetch the snapshot at record.transformation_history[0].data_shape
ref.
  2.2 Verify sha256 against stored hash. Mismatch = STOP, file under "snapshot integrity".
  2.3 Mount sources read-only.

STEP 3 — Replay lineage
  3.1 Run deterministic segments first. Assert byte-identical outputs against stored hashes.
  3.2 Run non-deterministic segments with N=10 trials.
  3.3 Record decision_origin distribution, retrieval set overlap, tool sequence.

STEP 4 — Compare
  4.1 Compute decision-path concordance against thresholds in this spec.
  4.2 If below threshold, mark as REGRESSION and route per s2-regression-discipline.
  4.3 If above threshold, mark as REPRODUCED.

STEP 5 — Record
  5.1 Emit a new provenance record with decision_origin=fallback if the re-run was
triggered by an incident, or decision_origin=agent if routine.
  5.2 Link to original record via corrections[].supersedes_record_id only if a correction
is being issued.

POSTCONDITIONS
[ ] Re-run outcome appended to record.corrections or to the regression log.
[ ] Reviewer signed off with timestamp + role.
[ ] Any pin loss filed as a separate reliability incident.
```

Operating Notes

- Re-runs are not free. Snapshot storage cost is real; retention class governs it.
- Vendor pin loss (a model revision disappearing) is the dominant real-world threat to reproducibility. Track vendor deprecation schedules in the Source Inventory.
- "I re-ran it and it looked the same" is not reproducibility. The runbook above is.