AXILOG

SRS-004

Inference Containment, Oracle Contract & L4/L5 Integration

Locked v0.3 L3 [D3]

Last updated 26 March 2026
Depends on SRS-001 · SRS-002 · SRS-003 · DVEC-001 · AXIOMA-FRAMEWORK
Changelog 1 entry

Contents

Depends on: SRS-001 v0.3 · SRS-002 v0.3 · SRS-003 v0.3 · DVEC-001 v1.3 · AXIOMA-FRAMEWORK v0.4

---

0. GOVERNING PRINCIPLE

> **LLMs are not trusted. They are recorded.**
>
> **Inference is not execution. Inference is evidence.**

If L4/L5/L6 are about enforcing determinism, L3 is about **containing non-determinism** without letting it leak.

We are not trying to make inference deterministic. We are making it:

- **Admissible** — captured and committed before use
- **Bounded** — constrained by schema, size, and domain
- **Replayable in effect** — downstream behaviour is deterministic
- **Provable in context** — every decision traceable to evidence

---

1. PURPOSE

This document defines the **Inference Containment Contract** for the Axioma framework.

It specifies how inherently non-deterministic systems (LLMs, ML models, retrieval systems, external APIs) are:

- admitted into Axioma
- bounded in behaviour
- made observable
- constrained by L4 policies
- anchored to L6 evidence

Additionally, this document defines:

- the **L5 → L4 integration contract** for policy enforcement

- the **Policy DSL specification** for deterministic rule definition
- the **cross-layer ordering invariants** binding L3, L4, L5, and L6

Objective: Non-determinism SHALL be contained, not eliminated, and SHALL NOT violate system determinism at L4/L5/L6.

---

2. DEFINITIONS

2.1 Oracle

An **oracle** is any external system producing non-deterministic output.

Examples:

- LLMs (GPT, Claude, Llama, etc.)
- ML models (classifiers, regressors, embedders)
- Retrieval systems (vector databases, search indices)
- External APIs (weather, market data, etc.)

2.2 AX:OBS:v1 (Oracle Observation)

All oracle outputs MUST be admitted as `AX:OBS:v1` records.

An admitted observation is:

- canonicalised input
- committed to the ledger
- immutable once admitted

2.3 Inference Event

An **inference event** is:

```
Input → Oracle → Output → Admission → Policy → Behaviour
```

2.4 Containment Boundary

The **containment boundary** is the point at which:

```
non-determinism → becomes immutable evidence
```

2.5 Policy DSL

The **Policy DSL** is a static definition format for safety rules, using RFC 8785 (JCS) canonical JSON, compiled into the policy registry at build time.

2.6 Certified Operational Envelope (COE)

The **COE** is the set of all states permitted by all active policies. Defined in SRS-003.

2.7 Completion State

The **completion state** indicates whether oracle output was fully received:

Value	Meaning
`COMPLETE`	Full output received
`TRUNCATED`	Output exceeded size bound
`ERROR`	Oracle invocation failed

2.8 Failure Type

The **failure type** classifies oracle invocation failures:

Value	Meaning
`NULL`	No failure
`TIMEOUT`	Oracle did not respond in time
`INVALID_OUTPUT`	Output failed schema validation
`TRANSPORT_ERROR`	Network or protocol failure

---

3. DETERMINISM MODEL

3.1 Determinism Class

L3 SHALL operate under:

D3 — Bounded Non-Deterministic

3.2 Containment Principle

SRS-004-SHALL-001

Oracle outputs SHALL NOT influence system behaviour unless:

1. admitted as `AX:OBS:v1`
2. validated by L4 policy evaluation
3. committed to L6 ledger

Verification: inspection, replay verification, integration test

3.3 Deterministic Downstream Guarantee

SRS-004-SHALL-002

Once admitted, all downstream behaviour SHALL be deterministic.

Given identical `AX:OBS:v1` sequences, the system SHALL produce identical:

- `AX:POLICY:v1` records
- `AX:TRANS:v1` records
- final state

Verification: replay verification, cross-platform identity test

4. ORACLE ADMISSION CONTRACT

4.1 Mandatory Admission

SRS-004-SHALL-003

All oracle outputs MUST be:

```
captured → canonicalised → committed (AX:OBS:v1)
```

before use.

Verification: inspection, static analysis, integration test

4.2 No Direct Use

SRS-004-SHALL-004

Oracle outputs SHALL NOT be used directly by:

- L5 (agent state machine)
- L4 (policy evaluator)

All oracle influence MUST flow through the admission → policy → transition pipeline.

Verification: static analysis, inspection

4.3 Admission Ordering

SRS-004-SHALL-005

Oracle admission SHALL occur in strict ledger order:

```
ORDER BY ledger_seq ASC
```

No out-of-order admission is permitted.

Verification: property test, concurrency test

4.4 Observation Ordering Guard

SRS-004-SHALL-038

Closure fix v0.2 — prevents replay/injection attacks via sequence regression.

Each new `AX:OBS:v1` SHALL satisfy:

```
ledger_seq_new > ledger_seq_last
```

Violation Behaviour:

If `ledger_seq_new ≤ ledger_seq_last`:

- violation SHALL be raised
- agent SHALL transition to `STOPPED`

This mirrors SRS-002-SHALL-010 (Time Oracle monotonicity) but applies to all oracle observations.

Verification: property test, fault injection test, state machine test

---

5. CANONICALISATION

5.1 Canonical Form

SRS-004-SHALL-006

All oracle outputs MUST be:

- serialised deterministically
- RFC 8785 (JCS) canonical

Verification: JCS linter, hash stability test

5.2 Required Fields

Each `AX:OBS:v1` (oracle observation) SHALL include:

Field	Type	Description
`completion_state`	enum	COMPLETE, TRUNCATED, or ERROR
`failure_type`	enum/null	NULL, TIMEOUT, INVALID_OUTPUT, or TRANSPORT_ERROR
`input_hash`	bytes32	SHA-256 of canonical input/prompt
`ledger_seq`	uint64	Ordering anchor
`model_id`	string	Model/version identifier
`obs_hash`	bytes32	SHA-256 of this canonical record (§5.7)
`oracle_id`	string	Identity of oracle
`output`	string	Canonicalised, normalised output
`output_size`	uint64	Byte length of output
`params`	object	Sampling parameters (see §5.4)
`schema_version`	string	Always “AX:OBS:v1”

Field Order: Fields MUST appear in lexicographic order per RFC 8785:

```
completion_state, failure_type, input_hash, ledger_seq, model_id,
obs_hash, oracle_id, output, output_size, params, schema_version
```

5.3 Forbidden Fields

The following SHALL NOT appear in `AX:OBS:v1`:

- wall-clock timestamps (unless separately admitted as Time Oracle)
- floating-point values
- non-canonical structures
- memory addresses
- process/thread IDs

Verification: schema test, inspection

5.4 Parameter Canonicalisation

SRS-004-SHALL-036

Closure fix v0.2 — eliminates hash divergence from parameter ordering.

The `params` object SHALL:

- follow RFC 8785 (JCS) canonicalisation
- use fixed field set in lexicographic order:

Field	Type	Description
`max_tokens`	uint32/null	Maximum output tokens
`seed`	uint64/null	Random seed if available
`temperature`	Q16.16/null	Sampling temperature
`top_p`	Q16.16/null	Nucleus sampling threshold

- include `null` for absent fields (no omission)

Canonical Format:

```
{"max_tokens":4096,"seed":null,"temperature":45875,"top_p":58982}
```

Note: `temperature` 45875 = 0.7 in Q16.16 (0.7 × 65536 ≈ 45875). Note: `top_p` 58982 = 0.9 in Q16.16 (0.9 × 65536 ≈ 58982).

Verification: JCS linter, hash stability test, cross-platform test

5.5 Encoding Canonicality

SRS-004-SHALL-042

Closure fix v0.2 — eliminates Unicode normalisation divergence.

All oracle inputs and outputs SHALL:

- be UTF-8 encoded
- use NFC (Canonical Decomposition, followed by Canonical Composition) normalisation form

Non-conformant encoding SHALL:

- be rejected prior to admission
- trigger `BREACH`

Verification: encoding test, property test

5.6 Output Normalisation

SRS-004-SHALL-043

Closure fix v0.3 — eliminates line ending and control character divergence.

Oracle output SHALL be normalised before admission:

- Line endings MUST be LF (`\n` , U+000A)
- CRLF (`\r\n`) MUST be converted to LF
- CR (`\r`) alone MUST be converted to LF
- Output MUST be valid UTF-8 (per SHALL-042)
- Control characters (U+0000–U+001F excluding U+000A) MUST be rejected

Rejection Behaviour:

If output contains forbidden control characters:

- `completion_state = ERROR`
- `failure_type = INVALID_OUTPUT`
- `BREACH` triggered

Verification: property test, encoding test, cross-platform test

5.7 String Escaping Canonicality

SRS-004-SHALL-045

Closure fix v0.3 — eliminates JSON escaping divergence.

All JSON string encoding SHALL use minimal escaping:

- Escape ONLY the following characters:
  - `"` (U+0022) → `\"`
  - `\` (U+005C) → `\\`
  - Control characters (U+0000–U+001F) → `\uXXXX`
- Direct UTF-8 encoding MUST be used for all other characters

- Unicode escape sequences (`\uXXXX`) MUST NOT be used where direct UTF-8 encoding is valid

Example:

```
CORRECT:   {"output":"Héllo wörld"}
FORBIDDEN: {"output":"H\u00e9llo w\u00f6rld"}
```

Verification: JCS linter, hash stability test, cross-platform test

5.8 Observation Hash

SRS-004-SHALL-046

Closure fix v0.3 — enables tamper detection and independent verification.

Each `AX:OBS:v1` record SHALL include an `obs_hash` field computed as:

```
obs_hash = SHA-256(canonical_record_without_obs_hash)
```

Computation Procedure:

1. Construct the complete `AX:OBS:v1` record with `obs_hash` set to the empty string `""`
2. Canonicalise per RFC 8785 (JCS)
3. Compute SHA-256 of the canonical bytes
4. Replace `obs_hash` with the computed hash (hex-encoded, lowercase)
5. Re-canonicalise (hash position is now populated)

Verification: hash stability test, property test

5.9 Schema Versioning

SRS-004-SHALL-047

Closure fix v0.3 — enables future schema evolution.

Each `AX:OBS:v1` record SHALL include:

```
"schema_version": "AX:OBS:v1"
```

This field:

- MUST be present in every observation record
- MUST be validated on admission
- enables forward-compatible schema evolution

Verification: schema test, inspection

---

6. OBSERVATION COMPLETENESS

6.1 Completeness Requirement

SRS-004-SHALL-035

Closure fix v0.2 — makes AX:OBS:v1 fully self-describing.

Every `AX:OBS:v1` SHALL include:

Field	Type	Description
`completion_state`	enum	COMPLETE, TRUNCATED, or ERROR
`failure_type`	enum/null	NULL, TIMEOUT, INVALID_OUTPUT, TRANSPORT_ERROR
`params`	object	Canonical sampling parameters (§5.4)

Invariant:

An auditor examining any `AX:OBS:v1` record SHALL be able to determine:

- Was this output complete or truncated?
- Did the oracle fail, and how?
- What sampling parameters were used?

Without reference to external context.

Verification: schema test, inspection

6.2 Maximum Observation Size

SRS-004-SHALL-048

Closure fix v0.3 — makes size bound explicit and verifiable.

The maximum size of an `AX:OBS:v1` record SHALL be:

```
#define AX_MAX_OBS_BYTES  65536  /* 64 KiB */
```

Any observation exceeding this limit:

- SHALL be recorded with `completion_state = TRUNCATED`
- SHALL trigger policy evaluation
- MAY be rejected as `BREACH` depending on policy

Verification: property test, boundary test

---

7. ORACLE IDENTITY

7.1 Identity Requirement

SRS-004-SHALL-007

Each oracle SHALL have:

- globally unique identifier (`oracle_id`)
- versioned model identity (`model_id`)

7.2 Stability Requirement

SRS-004-SHALL-008

Oracle identity MUST be:

- immutable for a given execution run
- identical across replay context

A mismatch between recorded `model_id` and replay environment SHALL trigger a verification failure.

Verification: inspection, replay verification

---

8. INPUT REPRODUCIBILITY

8.1 Prompt Determinism

SRS-004-SHALL-009

The input to the oracle SHALL be:

- fully specified (no implicit context)
- serialised deterministically
- hashed prior to execution

8.2 Hash Binding

```
input_hash = SHA-256(canonical_input)
```

The `input_hash` field in `AX:OBS:v1` cryptographically binds the observation to the exact input that produced it.

Verification: hash stability test, inspection

8.3 Input Schema Enforcement

SRS-004-SHALL-037

Closure fix v0.2 — closes silent replay breakage from prompt divergence.

Oracle inputs SHALL:

- conform to a declared input schema
- be canonicalised before hashing (RFC 8785 + NFC + line normalisation)
- be rejected if schema-invalid

Rationale: Two systems constructing slightly different prompts will produce different `input_hash` values, causing replay to fail silently. Schema enforcement ensures structural consistency.

Verification: schema test, property test, cross-platform test

9. NON-DETERMINISM BOUNDING

9.1 Output Bounding

SRS-004-SHALL-010

Oracle outputs SHALL be bounded by:

- maximum size (bytes) — see §6.2
- schema constraints
- allowed value domain

9.2 Schema Enforcement

SRS-004-SHALL-011

Outputs MUST conform to a declared schema:

```
schema → validated → admitted
```

Invalid outputs → `BREACH` (L4)

Verification: schema test, property test

9.3 Sampling Parameter Recording

SRS-004-SHALL-012

If the oracle uses stochastic sampling (temperature, top-p, etc.), all sampling parameters SHALL be recorded in the `AX:OBS:v1` record per §5.4.

Verification: inspection, schema test

9.4 Output Size Enforcement

SRS-004-SHALL-040

Closure fix v0.2 — prevents silent truncation from undermining replay.

If output exceeds maximum size:

- output SHALL NOT be truncated silently
- output SHALL be recorded with:
  - `completion_state = TRUNCATED`
  - `output_size` = actual byte length before truncation
- truncation SHALL trigger `BREACH` unless explicitly permitted by policy

Verification: property test, fault injection test

9.5 Atomic Output Requirement

SRS-004-SHALL-041

Closure fix v0.2 — prevents non-deterministic chunking from streaming.

Oracle outputs SHALL:

- be captured only after completion
- not be processed incrementally or streamed

Rationale: Streaming outputs produce non-deterministic chunking boundaries. Only complete outputs can be canonicalised deterministically.

Verification: inspection, integration test

---

10. ORACLE ISOLATION

10.1 Multi-Oracle Composition

SRS-004-SHALL-039

Closure fix v0.2 — prevents hidden coupling between oracle invocations.

Each oracle invocation SHALL:

- be independently admitted
- have its own `AX:OBS:v1` record
- not modify inputs of other oracle calls unless explicitly recorded as a new input

Chained Oracle Pattern:

If Oracle A output influences Oracle B input:

```
Oracle A → AX:OBS:v1 (A) → commit
            ↓
[construct input for B using A output]
            ↓
input_B = f(output_A)  ← this derivation MUST be deterministic
            ↓
Oracle B → AX:OBS:v1 (B) → commit
```

The derivation function `f` MUST be:

- deterministic (D1 or D2)
- traceable in evidence

Verification: inspection, integration test

---

11. POLICY GATING (L4 INTEGRATION)

11.1 Mandatory Evaluation

SRS-004-SHALL-013

All oracle outputs MUST be evaluated by L4 policies before influencing L5 behaviour.

11.2 Breach Behaviour

SRS-004-SHALL-014

If policy returns `BREACH`:

- output SHALL be rejected
- L5 SHALL transition to `ALARM` or `STOPPED`
- `AX:TRANS:v1` SHALL record the breach

Verification: integration test, state machine test

11.3 L5 → L4 Integration Contract

SRS-004-SHALL-015

The L5 agent step function SHALL:

1. Extract policy-relevant values from admitted `AX:OBS:v1`
2. Evaluate all policies in `policy_id` order (SRS-003-SHALL-012)
3. If ANY policy returns `BREACH`, override intended transition
4. Commit `AX:TRANS:v1` AFTER policy evaluation completes
5. Mutate state ONLY if commit succeeds

Verification: inspection, integration test, state machine test

11.4 Policy Binding

SRS-004-SHALL-044

Closure fix v0.3 — ensures explicit trace linkage between policy and observation.

Each `AX:POLICY:v1` record SHALL be unambiguously bound to exactly one `AX:OBS:v1` via `obs_ledger_seq`.

Required Field Addition to AX:POLICY:v1:

Field	Type	Description
`obs_ledger_seq`	uint64	Ledger sequence of the evaluated observation

Invariant:

For any `AX:POLICY:v1` record, the binding:

```
AX:POLICY:v1.obs_ledger_seq → AX:OBS:v1.ledger_seq
```

MUST be unambiguous and verifiable.

Rationale: Without explicit binding, batched or parallelised evaluation loses trace linkage. This field ensures every policy evaluation is traceable to the exact observation it evaluated.

Verification: property test, integration test, RTM generation

---

12. EXECUTION PIPELINE

12.1 Required Order

```
deterministic_input
→ oracle_execution
→ AX:OBS:v1 admission
→ AX:POLICY:v1 evaluation (0..N)
→ AX:TRANS:v1 behaviour (1)
```

12.2 Ordering Invariant

SRS-004-SHALL-016

For each inference event:

```
OBS → POLICY → TRANS
```

MUST hold.

This is the cross-layer ordering invariant from SRS-003-SHALL-022, extended to include L3.

Verification: property test, integration test

12.3 Multi-Layer Defensive Guard

The integrated system enforces three guards:

Layer	Guard	Mechanism
L4	Policy Guard	Policies evaluated in `policy_id` order using Q16.16 math
L5	State Guard	Any `BREACH` trapped before `AX:TRANS:v1` generation
L6	Ledger Guard	Transition cryptographically bound to ledger sequence

13. REPLAY MODEL

13.1 Replay Definition

SRS-004-SHALL-017

Replay SHALL:

- NOT re-execute oracle
- reuse recorded `AX:OBS:v1`

13.2 Replay Guarantee

SRS-004-SHALL-018

Given identical `AX:OBS:v1` sequence, system SHALL produce identical:

- policy outcomes (`AX:POLICY:v1`)
- behaviour transitions (`AX:TRANS:v1`)
- final state

Verification: replay verification, cross-platform identity test

13.3 Oracle Isolation

During replay:

- oracle is NOT invoked
- `AX:OBS:v1` records are read from ledger
- downstream processing is bit-identical

14. SIDE-EFFECT PROHIBITION

14.1 No Direct Effects

SRS-004-SHALL-019

Oracle outputs SHALL NOT:

- perform I/O
- mutate system state directly
- bypass L4/L5

Verification: static analysis, inspection

14.2 Effect Containment

All effects of oracle output MUST flow through:

```
AX:OBS:v1 → AX:POLICY:v1 → AX:TRANS:v1 → state mutation
```

---

15. FAILURE SEMANTICS

15.1 Oracle Failure

SRS-004-SHALL-020

If oracle invocation fails:

- SHALL produce failure observation
- SHALL be admitted as `AX:OBS:v1` with:
  - `completion_state = ERROR`
  - `failure_type` = appropriate failure classification
- SHALL trigger policy evaluation

15.2 Invalid Output

SRS-004-SHALL-021

Invalid outputs (schema violation, bounds exceeded) SHALL:

- be treated as `BREACH`
- force `STOPPED` if critical

15.3 Timeout Handling

SRS-004-SHALL-022

Oracle timeout SHALL:

- produce timeout observation
- be admitted as `AX:OBS:v1` with:
  - `completion_state = ERROR`
  - `failure_type = TIMEOUT`
- trigger policy evaluation (typically `BREACH`)

Verification: fault injection test, integration test

---

16. PURITY RESTORATION

16.1 Boundary Principle

SRS-004-SHALL-023

The system SHALL restore determinism at the containment boundary:

```
Non-deterministic (oracle) → admitted (AX:OBS:v1) → deterministic (L4/L5/L6)
```

16.2 Invariant Statement

> Once an oracle output crosses the containment boundary, the system is indistinguishable from a purely deterministic system processing immutable input data.

Verification: replay verification, property test

16.3 The Axioma Inference Theorem

> Non-deterministic systems can be used safely IF AND ONLY IF they are transformed into immutable, canonical, ordered evidence before influencing deterministic computation.

17. POLICY DSL SPECIFICATION

17.1 DSL Philosophy

The Axioma Policy DSL is not a runtime-interpreted script; it is a **Static Definition Format**. Policies are defined in JCS-canonical JSON and compiled into the `ax_policy_registry_t` at build time.

This ensures:

- zero dynamic allocation
- O(1) lookup
- bit-identical evaluation across platforms

17.2 DSL Schema Requirements

SRS-004-SHALL-024

The Policy DSL SHALL use RFC 8785 (JCS) for all rule definitions to ensure bit-identical parsing across platforms.

SRS-004-SHALL-025

Every policy definition MUST strictly follow the lexicographical field order: `comparison`, `enabled`, `policy_id`, `threshold`.

SRS-004-SHALL-026

Numeric thresholds MUST be defined as signed integers, which are converted to Q16.16 at compile-time.

Verification: JCS linter, build verification, cross-platform test

17.3 Canonical Policy Structure

A conformant DSL entry for a single safety rule:

```
{
  "comparison": "GT",
  "enabled": true,
  "policy_id": "POL-001-MAX-VELOCITY",
  "threshold": 4587520
}
```

Note: `4587520` is the Q16.16 representation of the integer `70` (70 × 65536 = 4587520).

17.4 Comparison Operators

SRS-004-SHALL-027

The DSL SHALL support a closed set of comparison operators:

Operator	Meaning
`GT`	Greater than
`LT`	Less than
`GE`	Greater than or equal
`LE`	Less than or equal

SRS-004-SHALL-028

Any operator outside this closed set MUST result in a `BREACH` during evaluation (Fail-Safe).

Verification: property test, schema test

17.5 Static Registry Mapping

DSL Field	C Implementation
`policy_id`	`const char *policy_id`
`threshold`	`q16_16_t threshold`
`comparison`	`ax_policy_cmp_t comparison`
`enabled`	`uint8_t enabled`

17.6 DSL Evaluation Invariants

When an input is processed:

1. **Retrieve:** Fetch rule from the registry
2. **Compare:** Execute pure function `ax_policy_check_breach()` using Q16.16 math
3. **Witness:** Generate `AX:POLICY:v1` record in lexicographical field order

---

18. FIXED-POINT ARITHMETIC COMPLIANCE

18.1 Determinism Fix

SRS-004-SHALL-029

Fixed-point conversion from integer SHALL use multiplication, not left-shift, to avoid C99 undefined behaviour on negative values.

```
/* CORRECT — defined for all signed integers */
#define Q16_16_FROM_INT(x)  ((q16_16_t)((x) * Q16_16_ONE))

/* FORBIDDEN — undefined for negative x in C99 */
#define Q16_16_FROM_INT(x)  ((q16_16_t)((x) << 16))
```

Rationale: In C99, left-shifting a negative value is undefined behaviour. Multiplication by the scale factor (2^16) is defined and deterministic across all conformant compilers.

Verification: static analysis (Wshift-negative-value), cross-platform test

18.2 Arithmetic Wrapper Requirement

SRS-004-SHALL-030

All policy arithmetic SHALL use DVM arithmetic wrappers:

```
✅ dvm_add_q16(), dvm_mul_q16(), dvm_div_q16()
❌ Raw operators (+, -, *, /) on fixed-point types
```

Verification: static analysis (AST-level), pre-commit guardrail

---

19. L5 AGENT INTEGRATION

19.1 Agent Step Contract

SRS-004-SHALL-031

The `ax_agent_step()` function SHALL implement the following phases:

Phase A — Policy Evaluation (L4)

- Extract policy-relevant values from input
- Evaluate all policies in deterministic order
- Capture aggregate result (`PERMITTED` or `BREACH`)

Phase B — Pre-Commit Invariant (L5/L6)

- Determine next state (considering policy result)
- Commit `AX:TRANS:v1` before mutation

Phase C — State Mutation

- Update internal health ONLY if commit succeeds
- If commit fails, transition to `STOPPED` (SRS-002-SHALL-025)

19.2 Policy Override Behaviour

SRS-004-SHALL-032

If policy evaluation returns `BREACH` :

- intended transition SHALL be overridden
- next state SHALL be `ALARM` or `STOPPED`
- breach SHALL be recorded in `AX:TRANS:v1`

Verification: integration test, state machine test

19.3 Terminal State Guard

SRS-004-SHALL-033

If agent is in `STOPPED` state:

- step function SHALL set protocol fault

- SHALL NOT proceed with evaluation
- SHALL return immediately

(Per SRS-002-SHALL-004)

Verification: state machine test, inspection

---

20. TRACEABILITY

20.1 Evidence Chain

Each inference event SHALL produce:

```
AX:OBS:v1 (oracle output)
→ AX:POLICY:v1 (0..N policy evaluations)
→ AX:TRANS:v1 (1 state transition)
```

20.2 Audit Completeness

SRS-004-SHALL-034

Every oracle output SHALL be traceable to:

- the input that triggered it (`input_hash`)
- the observation itself (`obs_hash`)
- the policies that evaluated it (`AX:POLICY:v1` records via `obs_ledger_seq`)
- the behaviour it influenced (`AX:TRANS:v1` record)

Verification: RTM generation, inspection

---

21. PHASE 4 CLOSURE CRITERIA

Phase 4 is complete when:

21.1 Containment

- [x] Oracle outputs are fully contained
- [x] No direct oracle influence on behaviour

21.2 Replay

- [x] Replay does not require oracle execution
- [x] Replay produces bit-identical downstream results

21.3 Policy Integration

- [x] Policy gating enforced for all oracle outputs
- [x] `BREACH` correctly overrides L5 transitions
- [x] Policy → Observation binding explicit (v0.3)

21.4 Evidence

- [x] Evidence chain complete (OBS → POLICY → TRANS)
- [x] All records JCS-canonical
- [x] Observation hash included (v0.3)
- [x] Schema version included (v0.3)

21.5 DSL

- [x] Policy DSL compiles to static registry
- [x] Cross-platform evaluation produces identical results

21.6 Arithmetic

- [x] No undefined behaviour in fixed-point operations
- [x] All arithmetic uses DVM wrappers

21.7 Observation Completeness (v0.2)

- [x] AX:OBS:v1 is fully self-describing
- [x] Completion state and failure type recorded
- [x] Sampling parameters canonicalised

21.8 Boundary Closure (v0.2)

- [x] Input schema enforcement
- [x] Output size enforcement (no silent truncation)
- [x] Observation ordering guard (monotonicity)
- [x] Oracle isolation (multi-oracle composition)
- [x] Atomic output requirement (no streaming)
- [x] Encoding canonicality (UTF-8 + NFC)

21.9 Canonicalisation Closure (v0.3)

- [x] Output normalisation (line endings, control chars)
- [x] String escaping canonicality (minimal escaping)
- [x] Policy binding explicit (obs_ledger_seq)
- [x] Maximum observation size defined

---

22. REQUIREMENT SUMMARY

ID	Requirement	Section
SRS-004-SHALL-001	Containment principle	3.2
SRS-004-SHALL-002	Deterministic downstream	3.3
SRS-004-SHALL-003	Mandatory admission	4.1
SRS-004-SHALL-004	No direct use	4.2
SRS-004-SHALL-005	Admission ordering	4.3
SRS-004-SHALL-006	Canonicalisation	5.1
SRS-004-SHALL-007	Oracle identity	7.1
SRS-004-SHALL-008	Identity stability	7.2
SRS-004-SHALL-009	Prompt determinism	8.1
SRS-004-SHALL-010	Output bounding	9.1
SRS-004-SHALL-011	Schema enforcement	9.2
SRS-004-SHALL-012	Sampling recording	9.3
SRS-004-SHALL-013	Policy gating	11.1
SRS-004-SHALL-014	Breach behaviour	11.2
SRS-004-SHALL-015	L5 → L4 integration	11.3
SRS-004-SHALL-016	Ordering invariant	12.2
SRS-004-SHALL-017	Replay definition	13.1
SRS-004-SHALL-018	Replay guarantee	13.2
SRS-004-SHALL-019	Side-effect prohibition	14.1
SRS-004-SHALL-020	Oracle failure	15.1
SRS-004-SHALL-021	Invalid output	15.2
SRS-004-SHALL-022	Timeout handling	15.3
SRS-004-SHALL-023	Purity restoration	16.1

ID	Requirement	Section
SRS-004-SHALL-024	DSL JCS compliance	17.2
SRS-004-SHALL-025	DSL field order	17.2
SRS-004-SHALL-026	DSL integer thresholds	17.2
SRS-004-SHALL-027	Comparison operator set	17.4
SRS-004-SHALL-028	Unknown operator breach	17.4
SRS-004-SHALL-029	Fixed-point conversion	18.1
SRS-004-SHALL-030	Arithmetic wrappers	18.2
SRS-004-SHALL-031	Agent step contract	19.1
SRS-004-SHALL-032	Policy override	19.2
SRS-004-SHALL-033	Terminal state guard	19.3
SRS-004-SHALL-034	Audit completeness	20.2
SRS-004-SHALL-035	Observation completeness	6.1
SRS-004-SHALL-036	Parameter canonicalisation	5.4
SRS-004-SHALL-037	Input schema enforcement	8.3
SRS-004-SHALL-038	Observation ordering guard	4.4
SRS-004-SHALL-039	Oracle isolation	10.1
SRS-004-SHALL-040	Output size enforcement	9.4
SRS-004-SHALL-041	Atomic output requirement	9.5
SRS-004-SHALL-042	Encoding canonicality	5.5
SRS-004-SHALL-043	Output normalisation	5.6
SRS-004-SHALL-044	Policy binding	11.4
SRS-004-SHALL-045	String escaping canonicality	5.7
SRS-004-SHALL-046	Observation hash	5.8
SRS-004-SHALL-047	Schema versioning	5.9

ID	Requirement	Section
SRS-004-SHALL-048	Maximum observation size	6.2

Total: 48 SHALL requirements

---

23. LAYER INTEGRATION SUMMARY

Layer	Role	SRS
L6	Truth — what happened	SRS-001
L5	Behaviour — what to do	SRS-002
L4	Admissibility — what is allowed	SRS-003
L3	Containment — what was observed	SRS-004

Cross-Layer Invariant:

```
L3 (oracle) → L6 (commit) → L4 (evaluate) → L5 (behave) → L6 (commit)
```

No layer may influence another without a corresponding evidence record.

---

24. SYSTEM PROPERTIES (FINAL)

After v0.3 closure:

Property	Status
Oracle containment	✔ Closed
Input determinism	✔ Closed
Output determinism (post-admission)	✔ Closed
Replay	✔ Closed
Multi-oracle safety	✔ Closed
Encoding stability	✔ Closed
Evidence completeness	✔ Closed
Canonical JSON escaping	✔ Closed (v0.3)
Policy → Observation binding	✔ Closed (v0.3)
Output normalisation	✔ Closed (v0.3)

25. IMPLEMENTATION REFERENCE

25.1 Constants (`include/axilog/limits.h`)

```
/* include/axilog/limits.h */

/**
 * DVEC: v1.3
 * DETERMINISM: D1 — Strict Deterministic
 *
 * SRS-004-SHALL-048: Maximum observation size
 */

#ifndef AXILOG_LIMITS_H
#define AXILOG_LIMITS_H

/**
 * @brief Maximum size of an AX:OBS:v1 record in bytes.
 * SRS-004-SHALL-048
 */
#define AX_MAX_OBS_BYTES  65536  /* 64 KiB */

#endif /* AXILOG_LIMITS_H */
```

25.2 Types Header (`include/axilog/types.h`)

```
/* include/axilog/types.h */

/**
 * DVEC: v1.3
 * DETERMINISM: D1 — Strict Deterministic
 * MEMORY: Zero Dynamic Allocation
 *
 * SRS-004-SHALL-029: Fixed-point conversion uses multiplication.
 */

#ifndef AXILOG_TYPES_H
#define AXILOG_TYPES_H

#include <stdint.h>

typedef int32_t q16_16_t;

#define Q16_16_ONE   (65536)
#define Q16_16_HALF  (32768)

/**
 * @brief Convert integer to Q16.16 fixed-point.
 *
 * FIX: Use multiplication instead of left-shift for signed integers.
 * In C99, shifting a negative value is Undefined Behavior.
 * Multiplication by the scale factor (2^16) is defined and deterministic.
 *
 * SRS-004-SHALL-029
 */
#define Q16_16_FROM_INT(x)  ((q16_16_t)((x) * Q16_16_ONE))

#endif /* AXILOG_TYPES_H */
```

25.3 Observation Types (`include/axilog/oracle.h`)

```
/* include/axilog/oracle.h */

/**
 * DVEC: v1.3
 * DETERMINISM: D3 — Bounded Non-Deterministic (oracle layer)
 * MEMORY: Zero Dynamic Allocation
 *
 * SRS-004-SHALL-035: Observation completeness
 * SRS-004-SHALL-036: Parameter canonicalisation
 * SRS-004-SHALL-046: Observation hash
 * SRS-004-SHALL-047: Schema versioning
 */

#ifndef AXILOG_ORACLE_H
#define AXILOG_ORACLE_H

#include <stdint.h>
#include "axilog/types.h"
#include "axilog/limits.h"

/**
 * @brief Schema version string for AX:OBS:v1 records.
 * SRS-004-SHALL-047
 */
#define AX_OBS_SCHEMA_VERSION  "AX:OBS:v1"

/**
 * @brief Completion state for oracle observations.
 * SRS-004-SHALL-035
 */
typedef enum {
    AX_COMPLETION_COMPLETE   = 0,
    AX_COMPLETION_TRUNCATED  = 1,
    AX_COMPLETION_ERROR      = 2
} ax_completion_state_t;

/**
 * @brief Failure type for oracle observations.
 * SRS-004-SHALL-035
 */
typedef enum {
    AX_FAILURE_NULL            = 0,
    AX_FAILURE_TIMEOUT         = 1,
    AX_FAILURE_INVALID_OUTPUT  = 2,
    AX_FAILURE_TRANSPORT_ERROR = 3
} ax_failure_type_t;
```

```
/**
 * @brief Canonical sampling parameters.
 * SRS-004-SHALL-036
 *
 * Field order: max_tokens, seed, temperature, top_p (lexicographic)
 * NULL represented by sentinel values.
 */
typedef struct {
    int32_t  max_tokens;   /* -1 = null */
    int64_t  seed;         /* -1 = null */
    q16_16_t temperature;  /* INT32_MIN = null */
    q16_16_t top_p;        /* INT32_MIN = null */
} ax_oracle_params_t;

#define AX_PARAMS_NULL_INT32  (-1)
#define AX_PARAMS_NULL_INT64  (-1LL)
#define AX_PARAMS_NULL_Q16    (INT32_MIN)

#endif /* AXILOG_ORACLE_H */
```

25.4 Policy Types (`include/axilog/policy.h`)

```
/* include/axilog/policy.h */

/**
 * DVEC: v1.3
 * DETERMINISM: D2 — Constrained Deterministic
 * MEMORY: Zero Dynamic Allocation
 *
 * SRS-004-SHALL-044: Policy binding via obs_ledger_seq
 */

#ifndef AXILOG_POLICY_H
#define AXILOG_POLICY_H

#include <stdint.h>
#include "axilog/types.h"

/**
 * @brief AX:POLICY:v1 record structure.
 * SRS-004-SHALL-044: obs_ledger_seq binds policy to observation
 *
 * Field order (lexicographic):
 *   actual, ledger_seq, obs_ledger_seq, policy_id, result, threshold
 */
typedef struct {
    q16_16_t    actual;          /* Observed value */
    uint64_t    ledger_seq;      /* Policy record sequence */
    uint64_t    obs_ledger_seq;  /* SRS-004-SHALL-044: bound observation */
    const char *policy_id;       /* Policy identifier */
    uint8_t     result;          /* 0 = PERMITTED, 1 = BREACH */
    q16_16_t    threshold;       /* Policy threshold */
} ax_policy_record_t;

#endif /* AXILOG_POLICY_H */
```

25.5 Agent Step Function (`src/agent.c`)

```
/**
 * @file agent.c (L5)
 * @brief Integration with L4 Policy Layer
 *
 * DVEC: v1.3
 * DETERMINISM: D2 — Constrained Deterministic
 * MEMORY: Zero Dynamic Allocation
 *
 * SRS-004-SHALL-015: L5 → L4 integration contract
 * SRS-004-SHALL-031: Agent step contract
 * SRS-004-SHALL-032: Policy override behaviour
 * SRS-004-SHALL-033: Terminal state guard
 * SRS-004-SHALL-038: Observation ordering guard
 * SRS-004-SHALL-044: Policy binding
 */

#include "axilog/agent.h"
#include "axilog/policy.h"

void ax_agent_step(
    ax_agent_ctx_t   *agent_ctx,
    ax_policy_ctx_t  *policy_ctx,
    const ax_input_t *input,
    ct_fault_flags_t *faults)
{
    ax_health_state_t next_state;
    ax_policy_input_t p_input;
    ax_policy_result_t p_result;

    /* SRS-004-SHALL-033: Terminal state guard */
    /* SRS-002-SHALL-004: Terminal behaviour */
    if (agent_ctx->health == AX_HEALTH_STOPPED) {
        faults->protocol = 1;
        return;
    }

    /* SRS-004-SHALL-038: Observation ordering guard */
    if (input->ledger_seq <= agent_ctx->last_seen_seq) {
        faults->protocol = 1;
        agent_ctx->health = AX_HEALTH_STOPPED;
        agent_ctx->local_failed_flag = 1;
        return;
    }

    /*
     * PHASE A: Policy Evaluation (L4)
     * SRS-003-SHALL-011: Map input to policy value.
```

```
     * SRS-004-SHALL-015: L5 → L4 integration.
     * SRS-004-SHALL-044: Policy binding via obs_ledger_seq.
     */
    if (input→type == AX_INPUT_LLM_OBS || input→type == AX_INPUT_TIME_OBS) {
        p_input.value = ax_agent_extract_value(input);
        p_input.ledger_seq = input→ledger_seq;
        p_input.obs_ledger_seq = input→ledger_seq; /* SRS-004-SHALL-044 */

        /* SRS-003-SHALL-012: Evaluate all policies in deterministic order */
        p_result = ax_policy_evaluate_all(policy_ctx, &p_input, faults);

        if (p_result == AX_POLICY_BREACH) {
            /* SRS-004-SHALL-032: BREACH forces safety transition */
            /* SRS-003-SHALL-007: Violation propagation */
            next_state = AX_HEALTH_ALARM;
        } else {
            next_state = ax_agent_determine_next(agent_ctx→health, input-
>type);
        }
    } else {
        next_state = ax_agent_determine_next(agent_ctx→health, input→type);
    }

    /*
     * PHASE B: Pre-Commit Invariant (L5/L6)
     * SRS-002-SHALL-007: Commit transition before mutation.
     * SRS-004-SHALL-016: Ordering invariant (OBS → POLICY → TRANS).
     */
    ax_agent_commit_transition(agent_ctx, input, next_state, faults);

    /*
     * PHASE C: State Mutation
     * Only update internal health if the ledger commit succeeded.
     */
    if (!ct_fault_any(faults)) {
        agent_ctx→health = next_state;
        agent_ctx→last_seen_seq = input→ledger_seq;
    } else {
        /* SRS-002-SHALL-025: Fail-safe priority override */
        agent_ctx→health = AX_HEALTH_STOPPED;
        agent_ctx→local_failed_flag = 1;
    }
}
```

26. FINAL STATEMENT

The inference layer SHALL:

> Contain non-determinism such that all observable system behaviour remains deterministic, auditable, and replayable.

Key Insight:

Most systems:

- let inference directly drive behaviour
- cannot replay decisions
- cannot prove what the model saw

Axioma provides:

- Model output → becomes immutable fact → then evaluated → then permitted to influence behaviour

System Property:

> The system cannot behave differently without producing different evidence.

The Axioma Inference Theorem:

> Non-deterministic systems can be used safely IF AND ONLY IF they are transformed into immutable, canonical, ordered evidence before influencing deterministic computation.

27. REVISION HISTORY

Version	Date	Author	Changes
0.1	2026-03-26	William Murray	Initial draft — Inference Containment (34 SHALL), L5→L4 integration, Policy DSL, fixed-point arithmetic fix
0.2	2026-03-26	William Murray	Audit closure — added SHALL-035 through SHALL-042. Total: 42 SHALL.
0.3	2026-03-26	William Murray	Final closure — added SHALL-043 (output normalisation), SHALL-044 (policy binding), SHALL-045 (string escaping), SHALL-046 (observation hash), SHALL-047 (schema versioning), SHALL-048 (max observation size). Total: 48 SHALL. Status: Audit-Frozen FINAL.

28. DOCUMENT APPROVAL

Role	Name	Date	Signature
Author	William Murray	2026-03-26	
Reviewer			
Approver			

Appendix A — Audit Record

v0.1 → v0.2

Finding	Resolution
AX:OBS:v1 under-specified (completion state, failure type)	§6 + SHALL-035
Canonical structure for params undefined	§5.4 + SHALL-036
No explicit input schema for oracle inputs	§8.3 + SHALL-037
Ledger sequence monotonicity not enforced at L3	§4.4 + SHALL-038
Multi-oracle composition undefined	§10 + SHALL-039
Output size bound not enforced deterministically	§9.4 + SHALL-040
No explicit “no streaming” rule	§9.5 + SHALL-041
Missing UTF-8 + NFC encoding requirement	§5.5 + SHALL-042

v0.2 → v0.3

Finding	Resolution
Output encoding domain ambiguous (line endings, control chars)	§5.6 + SHALL-043
Policy → Observation binding implicit (no explicit trace)	§11.4 + SHALL-044
JSON string escaping not canonicalised	§5.7 + SHALL-045
No observation hash for tamper detection	§5.8 + SHALL-046
No schema versioning for forward compatibility	§5.9 + SHALL-047
Maximum observation size not explicit	§6.2 + SHALL-048

Appendix B — Canonical AX:OBS:v1 Example


```
{
  "completion_state": "COMPLETE",
  "failure_type": null,
  "input_hash":
"a1b2c3d4e5f67890123456789012345678901234567890123456789012345678901234abcd",
  "ledger_seq": 42,
  "model_id": "gpt-4-turbo-2024-04-09",
  "obs_hash":
"fedcba9876543210fedcba9876543210fedcba9876543210fedcba9876543210",
  "oracle_id": "azure-openai-prod-westeurope",
  "output": "The answer is 42.\n",
  "output_size": 18,
  "params": {
    "max_tokens": 4096,
    "seed": null,
    "temperature": 45875,
    "top_p": 58982
  },
  "schema_version": "AX:OBS:v1"
}
```


Notes:

- Fields in lexicographic order per RFC 8785
- Line ending normalised to LF
- Direct UTF-8 encoding (no unnecessary escapes)
- Q16.16 values for temperature (0.7) and top_p (0.9)
- obs_hash computed over record with empty obs_hash field

SRS-004 v0.3 — Audit-Frozen FINAL William Murray · SpeyTech · March 2026 Patent GB2521625.0
