AXILOG

SRS-001

Axilog Software Requirements Specification

Production Gold | v0.3 | L6 | [D1]

Last updated 15 March 2026
Supersedes SRS-001 v0.2
Depends on DVEC-001 · DVM-SPEC-001
Changelog 1 entry

Contents

Scope: Axilog substrate core **Alignment:** DVEC-001 v1.3 · DVM-SPEC-001 v1.0-rc1 · Axioma v0.4

Revision History

Version	Status	Notes
v0.1	Superseded	Initial draft — core substrate requirements
v0.2	Superseded	Genesis requirement, physical I/O fault handling, input domain validation, toolchain provenance
v0.3	Final	Static stack depth bound (SRS-010-SHALL-006), configuration evidence binding (SRS-001-SHALL-007)

1. Purpose

This document defines the normative software requirements for Axilog.

It is the authoritative source of SHALL statements for:

- Deterministic execution substrate behaviour
- Evidence commitment behaviour
- Ledger behaviour
- Fault propagation behaviour
- Oracle boundary handling
- Cross-platform identity verification
- Initialization determinism
- Physical storage fault handling
- Input domain validation
- Toolchain provenance

Every public Axilog function MUST trace to at least one requirement in this document.

2. Conformance

A software component is conformant only if:

- It implements all applicable SHALL statements
- Each SHALL has a declared verification method
- All verification methods pass
- All associated DVEC-001 v1.3 constraints are satisfied

Non-conformance is a system integrity failure.

3. Requirement Format

Each requirement is identified as:

```
SRS-SECTION-SHALL-NNN
```

Example: `SRS-006-SHALL-001`

Each requirement includes:

- ID
- Statement
- Rationale (where applicable)
- Verification method

Allowed verification methods:

Method	Description
`test`	Deterministic unit test
`property test`	Property-based test over input domain
`static analysis`	Automated static analysis tool
`inspection`	Manual code or design review
`cross-platform harness`	certifiable-harness compatible golden reference
`replay verification`	Full execution replay with commitment comparison
`fault injection test`	Induced fault condition verification
`schema test`	Evidence schema conformance test
`RTM generation`	Requirements traceability matrix tooling

---

4. Global System Requirements

SRS-001-SHALL-001 — Contract Declaration

Every Axilog module SHALL declare its governing DVEC version in the module header.

Verification: inspection, static analysis

SRS-001-SHALL-002 — Determinism Class Declaration

Every Axilog module SHALL declare its determinism class as D1, D2, or D3 in its primary header.

Verification: inspection, static analysis

SRS-001-SHALL-003 — SRS Traceability

Every public Axilog function SHALL include one or more SRS requirement anchors in its declaration or header comment.

Verification: static analysis, RTM generation

SRS-001-SHALL-004 — No Orphan Code

No public Axilog function SHALL exist without a corresponding SHALL statement.

Verification: RTM generation, static analysis

SRS-001-SHALL-005 — No Deferred Correctness

Production code SHALL contain no TODO, FIXME, HACK, OPTIMIZE, placeholder, or commented-out implementation blocks.

Verification: pre-commit scan, CI scan

SRS-001-SHALL-006 — Deterministic Initialization (Genesis Requirement)

Introduced in v0.2. Closes initial state ambiguity — the system is deterministic from the first instruction.

The Axilog substrate SHALL define a deterministic initialization sequence such that:

```
Power-On-Reset (POR) → S₀

∀ platforms P₁, P₂:
  S₀(P₁) = S₀(P₂)
```

Where `S₀` is the canonical initial state and MUST be representable as an `AX:STATE:v1` commitment.

Constraints:

Initialization SHALL NOT depend on:

- Prior memory contents
- Hardware timing variance
- Uninitialised state

All state MUST be explicitly initialised to defined values before any computation proceeds.

Verification: replay verification from POR, cross-platform identity test, golden reference generation at initialization boundary

---

SRS-001-SHALL-007 — Configuration Evidence Binding

Introduced in v0.3. Closes configuration spoofing — the system personality is cryptographically bound to the ledger from genesis.

The system configuration manifest SHALL be committed as the first `AX:OBS:v1` record in the ledger, prior to any state transition or evidence record.

Constraints:

- No state mutation SHALL occur before the configuration manifest is committed
- The configuration manifest SHALL include all parameters that define system behaviour (determinism class declarations, oracle configurations, policy versions, platform identifiers)
- A ledger whose first record is not a configuration manifest commitment SHALL be rejected as non-conformant
- Configuration manifest commits SHALL be included in the genesis golden reference

Rationale: Without this requirement, a system could replay with a different configuration while presenting valid subsequent commitments — "personality spoofing". Binding configuration to the ledger genesis makes such attacks detectable from the first record.

Verification: inspection, replay verification from genesis, schema test confirming first record type, fault injection test for missing manifest

---

5. Fault Context Requirements

SRS-005-SHALL-001 — Persistent Fault Context

All runtime primitives capable of arithmetic, logical, state, or ledger failure SHALL accept a `ct_fault_flags_t *faults` parameter.

Verification: static analysis, inspection

SRS-005-SHALL-002 — Explicit Fault Propagation

A function detecting a fault SHALL record that fault in the supplied fault context before returning.

Verification: property test, inspection

SRS-005-SHALL-003 — No Silent Arithmetic Failure

Overflow, underflow, divide-by-zero, invalid saturation, and invalid narrowing SHALL set a fault flag.

Verification: property test, fault injection test

SRS-005-SHALL-004 — No errno

Runtime fault signalling SHALL NOT use errno, global variables, or thread-local side channels.

Verification: static analysis

SRS-005-SHALL-005 — Fail-Closed Terminality

Any non-zero terminal fault state SHALL cause the runtime to transition to FAILED prior to any further state mutation.

Verification: state-machine test, fault injection test

SRS-005-SHALL-006 — No Implicit Fault Clearance

Fault flags SHALL NOT be cleared implicitly by any runtime primitive.

Verification: property test, inspection

SRS-005-SHALL-007 — Reset Requirement

Recovery from terminal FAILED state SHALL require an explicit reset operation defined by the owning subsystem. The reset boundary (process restart, subsystem reinitialisation, or system-level reset) SHALL be declared in the subsystem SRS.

Verification: inspection, state transition test

---

6. Ledger Requirements

SRS-006-SHALL-001 — Strict Total Order

The ledger SHALL define a strictly total order over all committed events.

Verification: property test, concurrency test

SRS-006-SHALL-002 — Chain Extension Function

Ledger chain extension SHALL be computed as:

$$L_0 = \text{SHA-256}(\text{"AX:LEDGER:v1"} \parallel \text{commit}(e_0))$$
$$L_n = \text{SHA-256}(\text{"AX:LEDGER:v1"} \parallel L_{n-1} \parallel \text{commit}(e_n))$$

Verification: hash stability test, inspection

SRS-006-SHALL-003 — Chain Tag Separation

The ledger SHALL use `AX:LEDGER:v1` only as a chain tag and SHALL NOT use it as an evidence type identifier.

Verification: tag linter, static analysis

SRS-006-SHALL-004 — Append-Only Behaviour

Committed ledger entries SHALL be append-only.

Verification: inspection, state test

SRS-006-SHALL-005 — No In-Place Mutation

Committed ledger entries SHALL NOT be updated or deleted.

Verification: inspection, storage interface test

SRS-006-SHALL-006 — Sequence Monotonicity

Ledger sequence position SHALL increase monotonically by exactly one per accepted append.

Verification: property test

SRS-006-SHALL-007 — Chain Integrity Failure

Any detected discontinuity, recomputation mismatch, or invalid prior hash SHALL transition the system to FAILED.

Verification: fault injection test, replay verification

SRS-006-SHALL-008 — Constant Space Append

Ledger append operations SHALL execute without heap allocation in runtime paths.

Verification: static analysis, allocator scan

SRS-006-SHALL-009 — Physical Storage Fault Handling (I/O Oracle)

Introduced in v0.2. Closes the physical reality gap — hardware failure is formally part of the model.

Any hardware-level I/O failure during ledger append SHALL:

1. Be captured as an oracle failure
2. Be recorded in fault context (`ct_fault_flags_t`)
3. Prevent partial or ambiguous chain mutation
4. Transition the system to FAILED prior to returning

Includes:

- Disk write failure
- fsync failure
- Partial write
- Timeout / device error
- Corrupted read-back verification

Constraints:

- No ledger state may exist in an indeterminate state
- No partial append may be considered committed

Verification: fault injection at storage layer, replay verification after induced I/O failure, property test: no valid chain continuation after failure

---

7. Evidence Requirements

SRS-007-SHALL-001 — Evidence Completeness

Every state transition influencing canonical state SHALL be preceded by a committed evidence record.

Verification: replay verification, inspection

SRS-007-SHALL-002 — Canonical JSON

All JSON evidence payloads SHALL be canonicalised using RFC 8785 before hashing.

Verification: JCS linter, hash stability test

SRS-007-SHALL-003 — Canonical Hash Input

Evidence commitments SHALL be computed on canonical byte representation, not raw structs or serializer-dependent output.

Verification: inspection, static analysis

SRS-007-SHALL-004 — Domain-Separated Commitment

Evidence commitments SHALL use the domain-separated function:

```
commit(e) = SHA-256(tag || LE64(|payload|) || payload)
```

Verification: hash test, inspection

SRS-007-SHALL-005 — Closed Evidence Tag Registry

Evidence type identifiers SHALL be limited to the DVEC domain separation registry.

Verification: tag linter

SRS-007-SHALL-006 — Evidence Version Closure

A v1 evidence type SHALL NOT change meaning without a formal version upgrade.

Verification: inspection, schema regression test

SRS-007-SHALL-007 — Required Evidence Types

The system SHALL support, at minimum:

- `AX:STATE:v1`
- `AX:TRANS:v1`
- `AX:OBS:v1`
- `AX:POLICY:v1`
- `AX:PROOF:v1`

Verification: inspection, schema presence test

SRS-007-SHALL-008 — Input Domain Validation

Introduced in v0.2. Closes the undefined input space — all inputs are contractually bounded before arithmetic or state mutation proceeds.

All public APIs accepting fixed-point or domain-constrained inputs SHALL:

1. Validate that inputs lie within the declared domain
2. Reject or fault any out-of-range input BEFORE:
   - Arithmetic execution
   - State mutation
   - Ledger commitment

Examples:

- Q16.16 range bounds
- Non-negative constraints where required
- Bounded parameter domains

Constraints:

- No implicit clamping without fault signalling
- No undefined behaviour from invalid inputs

Verification: property-based testing (domain boundaries), fault injection tests, static analysis for missing guards

---

8. Oracle Boundary Requirements

SRS-008-SHALL-001 — Oracle Admission

No oracle-derived value SHALL influence canonical state unless captured, canonicalised, committed, and referenced.

Verification: replay verification, inspection

SRS-008-SHALL-002 — Time as Oracle

Time SHALL be treated as an oracle input, not a runtime ambient value.

Verification: static analysis, inspection

SRS-008-SHALL-003 — No System Clock Access

Domain execution code SHALL NOT read wall clock or monotonic clock APIs directly.

Verification: pre-commit scan, CI scan

SRS-008-SHALL-004 — Time Observation Record

All accepted time injections SHALL be committed as `AX:OBS:v1` records with oracle type `TIME_INJECTION`.

Verification: schema test, replay verification

SRS-008-SHALL-005 — Monotonic Time

Time injections SHALL be monotonically non-decreasing unless explicitly configured otherwise at system initialisation.

Verification: property test, replay verification

SRS-008-SHALL-006 — Rollback Rejection

Undeclared time rollback SHALL be rejected and SHALL set terminal fault state.

Verification: fault injection test

SRS-008-SHALL-007 — D3 Routing

Observed non-deterministic components SHALL route output through the oracle boundary before state consumption.

Verification: inspection, architecture test

---

9. Arithmetic Requirements

SRS-009-SHALL-001 — Fixed-Point Only

All deterministic runtime arithmetic SHALL use fixed-point representation.

Verification: static analysis, forbidden pattern scan

SRS-009-SHALL-002 — Minimum Precision

The canonical fixed-point format SHALL be Q16.16 or stricter where declared.

Verification: inspection, type test

SRS-009-SHALL-003 — Widen Before Operation

All fixed-point multiplication, accumulation, or narrowing-sensitive operations SHALL widen operands before operation.

Verification: static analysis, inspection

SRS-009-SHALL-004 — Saturating Narrowing

All narrowing from widened intermediate representation SHALL saturate to target width and set a fault flag on saturation.

Verification: property test, fault injection test

SRS-009-SHALL-005 — No Raw Arithmetic on Fixed-Point Types

Runtime code SHALL NOT use raw `+`, `-`, `*`, `/` operators on declared fixed-point domain types outside approved primitives.

Verification: static analysis, AST linter

SRS-009-SHALL-006 — Deterministic Division

All fixed-point division primitives SHALL define rounding and fault behaviour explicitly.

Verification: property test

SRS-009-SHALL-007 — Zero Divide Fault

Division by zero SHALL set a terminal fault condition.

Verification: fault injection test

---

10. Memory Requirements

SRS-010-SHALL-001 — Zero Dynamic Allocation

Runtime execution paths SHALL perform no dynamic allocation.

Verification: static analysis, forbidden pattern scan

SRS-010-SHALL-002 — Caller-Owned Buffers

All runtime buffers SHALL be caller-provided or statically allocated at initialisation.

Verification: inspection, API review

SRS-010-SHALL-003 — Explicit Ownership

Every public API with pointer parameters SHALL declare memory ownership and lifetime semantics.

Verification: header inspection, static analysis

SRS-010-SHALL-004 — No Hidden Ownership Transfer

A function SHALL NOT assume ownership of caller memory unless explicitly declared in its contract.

Verification: inspection

SRS-010-SHALL-005 — Constant-Space Runtime

Core runtime primitives SHALL execute in bounded space without allocator dependence.

Verification: WCET/static resource analysis

SRS-010-SHALL-006 — Static Stack Depth Bound

Introduced in v0.3. Closes the stack overflow gap — forbidden dynamic allocation is necessary but not sufficient; stack depth must also be formally bounded.

The maximum call stack depth of all D1 runtime paths SHALL be statically determinable and SHALL NOT exceed the declared stack budget for the target platform.

Constraints:

- All call graphs SHALL be acyclic (no recursion, per DVEC §2.2)
- Maximum stack frame size per function SHALL be statically analysable
- Total worst-case stack depth SHALL be declared in the subsystem SRS and verified against platform constraints
- Stack overflow SHALL be treated as a terminal fault condition

Verification: static call graph analysis, stack depth analysis (e.g. clang-tidy stack usage plugin, GCC `-fstack-usage`), WCET tool integration

---

11. Cross-Platform Identity Requirements

SRS-011-SHALL-001 — Cross-Architecture Identity

D1 components SHALL produce bit-identical outputs across all supported architectures.

Verification: cross-platform harness

SRS-011-SHALL-002 — Supported Platforms

At minimum, cross-platform identity SHALL be verified across x86_64 and ARM64. RISC-V SHALL be added when supported.

Verification: harness matrix

SRS-011-SHALL-003 — Golden Reference Format

D1 systems SHALL emit a certifiable-harness compatible 368-byte golden reference.

Verification: golden verifier

SRS-011-SHALL-004 — Golden Match Requirement

A golden reference mismatch across supported architectures SHALL fail CI.

Verification: CI harness

SRS-011-SHALL-005 — Sequential and Parallel Equivalence

Any conformant parallel implementation SHALL produce bit-identical canonical state to the sequential reference implementation.

Verification: equivalence test

---

12. Concurrency Requirements

SRS-012-SHALL-001 — Deterministic Reduction Topology

Parallel reductions SHALL use a fixed tree topology.

Verification: inspection, equivalence test

SRS-012-SHALL-002 — No Shared Unsynchronised Mutation

Runtime code SHALL NOT use shared mutable state without explicit synchronisation and determinism proof.

Verification: static analysis, inspection

SRS-012-SHALL-003 — No Race-Dependent Semantics

Observable runtime behaviour SHALL NOT depend on race timing or scheduling coincidence.

Verification: concurrency test, inspection

SRS-012-SHALL-004 — Sequential Equivalence

Parallel execution SHALL preserve the canonical result of sequential execution.

Verification: equivalence test

---

13. Verification Infrastructure Requirements

SRS-013-SHALL-001 — Mechanised RTM

The system SHALL maintain a mechanised requirements traceability matrix linking SHALL statements to code anchors and verification artifacts.

Verification: RTM generation

SRS-013-SHALL-002 — Tag Linter

The build SHALL include a domain tag linter enforcing DVEC registry conformance.

Verification: CI execution

SRS-013-SHALL-003 — JCS Linter

The build SHALL include a JCS verification tool for all JSON evidence generation paths.

Verification: CI execution

SRS-013-SHALL-004 — Forbidden Pattern Gate

The build SHALL fail if forbidden constructs defined by DVEC are detected.

Verification: CI execution

SRS-013-SHALL-005 — Static Analysis Cleanliness

Static analysis SHALL pass with zero warnings for production code.

Verification: clang-tidy, cppcheck

SRS-013-SHALL-006 — Replay Verification

The system SHALL provide replay verification sufficient to prove evidence-order and commitment continuity.

Verification: replay verifier

SRS-013-SHALL-007 — Toolchain Provenance (Closure Invariant)

Introduced in v0.2. Closes meta-layer trust — even verification is verifiable.

All mechanised verification tools SHALL:

1. Be version-pinned
2. Produce bit-identical outputs for identical inputs
3. Be included within the certification boundary

Applies to:

- RTM generator

- JCS canonicaliser
- Static analysis tools
- Arithmetic linters
- Tag validators
- Golden verifier

Constraints:

- Tool version drift SHALL be treated as a system change
- No unverified tool upgrade permitted

Verification: toolchain hashing, reproducibility tests, cross-platform tool output comparison

---

14. Header Template Requirements

SRS-014-SHALL-001 — Compliance Header

Every public header SHALL declare:

- File purpose
- DVEC version
- Determinism class
- Memory model

Verification: static analysis, inspection

SRS-014-SHALL-002 — Requirement-Anchored Public API

Every public function declaration SHALL carry one or more SRS anchors.

Verification: RTM generation

Example:

```
/**
 * @brief Appends a new evidence commitment to the total-ordered chain.
 *
 * DVEC: v1.3
 * DETERMINISM: D1 — Strict Deterministic
 * MEMORY: Zero Dynamic Allocation
 *
 * SRS-006-SHALL-001: The ledger SHALL define a strictly total order.
 * SRS-006-SHALL-002: Chain extension SHALL use SHA-256("AX:LEDGER:v1"
 *                    || L_n-1 || commit(e_n)).
 * SRS-006-SHALL-009: Physical I/O failure SHALL transition to FAILED.
 */
void ax_ledger_append(
    ax_ledger_ctx_t  *ctx,
    const uint8_t     commit[32],
    ct_fault_flags_t *faults
);
```

---

15. Initial Public API Requirement Set

These are the first functions that SHALL exist only once their supporting SHALLs are frozen:

Function	Primary SRS Coverage
`dvm_add_q16`	SRS-009-SHALL-001..007
`dvm_sub_q16`	SRS-009-SHALL-001..007
`dvm_mul_q16`	SRS-009-SHALL-001..007
`dvm_div_q16`	SRS-009-SHALL-001..007
`ax_commit_evidence`	SRS-007-SHALL-001..008
`ax_ledger_append`	SRS-006-SHALL-001..009
`ax_verify_chain`	SRS-006-SHALL-007
`ax_time_observe`	SRS-008-SHALL-002..006
`ax_fault_reset`	SRS-005-SHALL-007 (only once reset semantics formally declared)

No additional public API SHALL be added without corresponding SHALL coverage.

16. Verification Matrix

Requirement ID	Function(s)	Verification
SRS-001-SHALL-006	initialization sequence	replay, golden reference
SRS-005-SHALL-003	`dvm_add_q16`, `dvm_mul_q16`, `dvm_div_q16`	property test, fault injection
SRS-005-SHALL-005	all faulting primitives	state machine test
SRS-006-SHALL-002	`ax_ledger_append`	hash stability test
SRS-006-SHALL-009	`ax_ledger_append`	fault injection, storage layer
SRS-007-SHALL-004	`ax_commit_evidence`	inspection, hash test
SRS-007-SHALL-008	all domain-constrained APIs	property test, fault injection
SRS-008-SHALL-005	`ax_time_observe`	replay verification, property test
SRS-011-SHALL-003	harness emitter	golden verifier
SRS-013-SHALL-007	all verification tools	toolchain hashing

17. Closure Assessment

Layer	Status
Execution determinism	✅ Closed
Fault propagation	✅ Closed
Ledger correctness	✅ Closed
Evidence system	✅ Closed
Oracle boundary	✅ Closed
Arithmetic model	✅ Closed
Memory model	✅ Closed
Cross-platform identity	✅ Closed
Initialization	✅ Closed (v0.2)
Physical I/O reality	✅ Closed (v0.2)
Input domain correctness	✅ Closed (v0.2)
Toolchain trust	✅ Closed (v0.2)
Stack depth bounds	✅ Closed (v0.3)
Configuration spoofing	✅ Closed (v0.3)

Certification Boundary:

Layer	Closing Requirement	Verification Tool
Meta-Layer	Toolchain Provenance (SRS-013-SHALL-007)	SHA-256 hashing of all tool binaries
Logic-Layer	Input Domain Validation (SRS-007-SHALL-008)	Property-based boundary testing
Physical-Layer	I/O Oracle (SRS-006-SHALL-009)	Storage layer fault injection
Stack-Layer	Static Stack Depth Bound (SRS-010-SHALL-006)	Static call graph + `-fstack-usage`
Genesis-Layer	Configuration Evidence Binding (SRS-001-SHALL-007)	Replay verification from genesis

Invariants of Truth:

1. **Temporal Invariant** — Time is an external oracle; rollback is a terminal failure
2. **Arithmetic Invariant** — Floating point is prohibited; all operations are widened and saturated with explicit fault signalling
3. **Memory Invariant** — Zero heap dependence ensures O(1) space complexity and eliminates fragmentation risks
4. **Reality Invariant** — Hardware failure is not a side effect; it is a formally modelled state transition to FAILED

---

18. Final Statement

Axilog SHALL not be a “mostly reproducible” runtime.

It SHALL be a deterministic, auditable, bounded execution substrate in which:

- Faults are explicit
- Evidence is canonical
- Order is total
- Cross-platform behaviour is identical
- Correctness is mechanically enforced

---

19. Closing Principle

> The purpose of this SRS is not to describe intentions. It is to define the exact set of behaviours the substrate is permitted to have.

SRS-001 v0.3 — Audit-Frozen William Murray · SpeyTech · March 2026 Patent GB2521625.0
