Network Working Group
Internet-Draft
Intended status: Informational
Expires: 15 September 2026

O. Romanchuk
Independent

14 March 2026

APSIX: Minimal Primitive Contract for Portable Semantic Runtimes
draft-romanchuk-apsix-00

Abstract

This document proposes *APSIX* (_Agent Portable Semantic Interface eXtension_) as a minimal primitive contract for portable semantic runtimes. The goal is not to standardize agent personalities, prompt formats, planner graphs, or model-specific reasoning behavior. The goal is to standardize the smallest useful contract of the runtime environment itself.

The design reflects runtime patterns observed in early governed semantic-runtime implementations. In those implementations, the primary managed resource is bounded semantic territory, and spawned actor populations are the main source of pressure on that resource. APSIX defines the lower-layer contract required for such runtimes to become portable across implementations.

The contract is intentionally narrow. It standardizes logical objects, lifecycle primitives, membrane-mediated decisions, capability semantics, event and provenance requirements, replay obligations, and progressive compliance profiles. It does not standardize model internals, scheduling strategies, storage engines, user interfaces, or application-specific orchestration logic.

The document is written to stand on its own as a self-contained runtime specification, without dependency on unpublished external framing.

This is a draft specification, not a final standard. The current name is provisional; the technical target is a portable primitive surface for governed semantic runtimes.

This document is intended as a *research specification*, not as an immediate industry standardization proposal. Its purpose is to define a portability target early enough that emerging runtime implementations do not harden around mutually incompatible framework-local abstractions before a shared primitive vocabulary exists.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 15 September 2026.

Table of Contents

1. Introduction

Agent runtimes are currently fragmented across framework-local abstractions. One system centers graphs, another centers conversations, another centers workflow automation, and another centers tool loops around a planner-worker core. This fragmentation weakens portability because applications depend on framework-specific object models rather than on a stable environment contract.

The design intent of APSIX is analogous in spirit to the role POSIX played for operating systems: not to prescribe every implementation detail, but to standardize a small enough primitive surface that higher-level software can become portable across runtimes.

The analogy to POSIX is aspirational in scope, not historical in maturity. The present ecosystem is still early. The purpose of this document is therefore not to claim that semantic runtimes have already converged enough for final standardization, but to provide a research-grade contract that implementations can target, compare against, and refine.

The conceptual shift is:

* classical POSIX standardized process, file, and I/O primitives;
* APSIX standardizes zone, membrane, artifact, and replay primitives for semantic runtimes.

The spec is derived from three commitments already stabilized in early governed semantic-runtime experimentation:

1. the runtime manages *semantic territory* rather than only tasks or model calls;
2. expansion pressure comes from *spawned actor populations*;
3. governability depends on *non-bypassable membrane mediation*, *durable artifact admission*, and *replayable authoritative state*.

APSIX treats control of spawn-driven semantic epidemics as a core runtime concern. This document does not yet mandate one universal epidemic-control algorithm for all runtimes or all partition regimes. Instead, APSIX standardizes the governance surface and invariants through which within-partition and cross-partition expansion must be admitted, bounded, observed, and reconstructed. Different runtimes may implement this surface through single-writer regimes, bounded speculative swarms, hierarchical descendant spawning, merge selection, quarantine, culling, or other control strategies, provided those strategies remain membrane-governed, budget-visible, locality-scoped, and replayable.

1.1. Scope

This document defines:

* a core object model for portable semantic runtimes;
* a minimal primitive set derived from runtime lifecycle;
* capability permissions and locality semantics;
* membrane and budget semantics;
* cross-boundary effect admission semantics;
* event, provenance, and replay requirements;
* compliance profiles.

This document does not define:

* prompt formats or chain-of-thought conventions;
* agent reasoning methods;
* model selection and vendor APIs;
* storage backends;
* scheduling algorithms;
* UI conventions;
* planner-specific orchestration graphs.

1.2. Intended Use

APSIX is intended to serve as:

* a portability target for experimental semantic runtime implementations;
* a comparison layer across framework-local runtimes;
* a basis for future conformance tests around membrane, provenance, replay behavior, and lifecycle-trace invariants.

It is not yet intended to serve as:

* a finalized industry interoperability standard;
* a replacement for existing agent SDK APIs;
* a mandate on internal scheduler or storage architecture.

1.3. Design Goals

The design goals are:

1. *Portability*: applications should rely on runtime primitives that survive framework changes.
2. *Governability*: the runtime must expose an enforcement locus for spawn, refinement, anchoring, and cross-boundary effects.
3. *Observability*: the runtime must emit stable records of semantic expansion and state transitions.

4. *Minimality*: the primitive set should be small enough to implement and reason about.
5. *Replayability*: authoritative state must be reconstructible without reproducing original latent reasoning traces.
6. *Extensibility*: richer runtimes should be able to add features without breaking core semantics.
7. *Epidemic Control Extensibility*: the contract should support multiple partition-local spawn-control algorithms without losing governability or replay semantics.

2. Conformance Language and Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Core Terms

Domain: a bounded semantic representation that supports localization, refinement, anchoring, and harvest.

Zone: the runtime embodiment of a domain, including state, policy, budgets, partitions, and artifacts.

Partition: a bounded sub-zone scope that defines the default unit of locality and refinement.

Actor: a spawned runtime instance operating under a capability mask and budget share.

Artifact: a produced object that can be retained, validated, rejected, anchored, or harvested.

Anchor: the promotion of an artifact into durable authoritative runtime state.

Membrane: the enforcement boundary around a zone and its sensitive state transitions.

Budget: an explicit bound on expansion, actor creation, artifact volume, validation cost, or other governed quantities.

Run: a concrete execution attempt or recovery/harvest attempt associated with an actor or task scope and having a terminal outcome.

Replay: authoritative reconstruction of zone state from events, anchors, and checkpoints; not reproduction of token-level reasoning.

Harvest: a runtime operation that returns anchored outputs from a zone.

Effect: an external or cross-boundary action whose primary result is not only a local artifact, but a mutation, publication, delivery, or other action against another authority boundary.

Spawn Regime: an implementation-defined policy governing the propagation dynamics of actor creation within a zone or partition, including concurrency limits, descendant spawning rules, competition, throttling, quarantine, merge, or termination behavior.

Ledger Manifest: a stable description of the replay-relevant ledger surfaces for a zone, including schema versions, runtime profile, measurement convention, and record paths.

3. Object Model

An APSIX-compliant runtime MUST expose the following logical objects.

These objects define logical runtime state, not a mandated storage substrate. Implementations MAY represent them using graphs, databases, event streams, logs, or other internal structures, provided the normative semantics of the contract remain observable.

3.1. Zone

A zone MUST have at least:

* zone_id
* domain_spec
* lifecycle_state
* membrane_policy_version
* budget_state
* partition_state
* authoritative_state_ref

In APSIX-governed and APSIX-replayable, a zone SHOULD additionally expose:

* output_boundary_ref or equivalent authoritative harvest target surface
* ledger_state
* ledger_manifest_ref, if a separate manifest object is used

Optional implementation fields MAY include ownership metadata, external bindings, topology hints, or storage references.

authoritative_state_ref denotes the runtime's single authoritative execution state for the zone. Implementations MAY expose dashboards, plans, manifests, or other operator-facing projections, but such projections MUST NOT function as independent control planes with authority to diverge from authoritative zone state.

3.2. Partition

A partition MUST have at least:

* partition_id
* zone_id
* scope_ref
* status

The runtime MAY materialize partitions explicitly or derive them from a structured domain substrate, but partition identity and locality semantics MUST be observable.

3.3. Actor

An actor MUST have at least:

* actor_id
* zone_id
* admitted_partitions
* capability_mask
* budget_share
* intent
* status

An actor MAY be ephemeral or persistent.

3.4. Run

A run MUST have at least:

* run_id
* zone_id
* request_id
* run_kind
* status
* started_at
* ended_at or equivalent terminal timestamp

If the runtime associates runs with an actor or partition-local task, it MUST expose that binding.

Run kinds MUST distinguish at minimum:

* execution_run
* recovery_run, if recovery is supported
* harvest_run, if harvest materializes execution-like work

3.5. Artifact

An artifact MUST have at least:

* artifact_id or provisional identifier
* origin_actor_id
* zone_id
* artifact_type
* status
* provenance

Artifact statuses MUST distinguish at least:

* generated
* anchored
* harvested
* rejected

3.6. Anchor Record

An anchored artifact MUST have an associated anchor record containing at least:

* anchor_id
* artifact_id
* zone_id
* policy_version
* decision
* timestamp_or_order

3.7. Membrane Decision Record

A membrane-mediated decision MUST have an auditable decision record containing at least:

* decision_id
* request_id
* zone_id
* request_type

* subject_ref
* decision
* policy_version
* reason_code
* capability_basis
* budget_context
* timestamp_or_order

3.8. Membrane Policy

A membrane policy MUST define:

* spawn admissibility rules
* within-partition population-control rules, if multiple actors per partition are supported
* refinement admissibility rules
* cross-zone and cross-partition effect rules
* external effect admission rules
* artifact promotion rules
* escalation rules
* budget interaction rules
* epidemic-control rules such as throttling, quarantine, culling, merge, or equivalent growth-control behavior, if such controls are supported

3.9. Event

The runtime MUST produce an event stream or event log containing at least:

* zone creation
* actor admission request and decision
* refinement event
* execute completion or failure
* external effect request, decision, completion, or failure, if such effects are supported
* anchor decision
* harvest event
* freeze or closure event

Each event record MUST expose at least:

* event_id
* seq_no or equivalent global authoritative order index
* zone_id
* event_type
* subject_ref

* request_id, if the event is part of a membrane-mediated request chain
* run_id, if the event is part of a concrete execution attempt
* timestamp_or_order
* causal_ref or equivalent parent reference, if such ordering is supported

If a runtime stores events, decisions, and anchors in separate ledgers, those ledgers MUST still preserve one authoritative cross-ledger ordering surface such as a shared monotonic seq_no.

3.10. Ledger Manifest

APSIX-replayable runtimes SHOULD expose a ledger manifest containing at least:

* zone_id
* schema_version
* runtime_profile
* measurement_convention, if telemetry proxies are exposed
* replay-relevant record paths or references
* record schema version or equivalent compatibility indicator

4. Primitive Set

APSIX-core runtimes MUST implement the lifecycle primitives in Sections 4.1-4.7. Governed and replayable profiles MUST additionally implement the observability primitive in Section 4.8.

4.1. zone(domain_spec) -> zone_id

Creates a zone from a domain specification.

Requirements:

* MUST initialize membrane and budget state.
* MUST initialize partition state.
* If the runtime exposes harvested outputs, MUST define an authoritative output boundary or equivalent harvest target surface for the zone.
* SHOULD bind or snapshot the authoritative output boundary for later harvest validation.
* MUST return a stable zone_id.
* MUST emit a zone creation event.

4.2. spawn(capability_set, zone_id, intent) -> actor_id | error

Requests creation of an actor in a zone.

Requirements:

* Each invocation of spawn() MUST admit at most one actor. Multi-actor populations MUST arise through repeated or descendant invocations of spawn(), not through batch return of a single call.
* MUST assign admitted partition scope or return an explicit placement failure.
* MUST assign a capability mask and budget share.
* MUST fail deterministically when policy or budget denies creation.
* MUST emit both request and decision events.
* In APSIX-governed and APSIX-replayable, spawn request and decision surfaces MUST be correlated by a stable request_id.
* If actor execution later materializes, the runtime MUST expose a run_id or equivalent execution identifier for that concrete attempt.
* A spawned actor operating within an admitted partition MAY decline further expansion, request refinement, or request descendant spawn under an implementation-defined Spawn Regime.
* If the runtime supports multiple actors within one partition or descendant spawning from within a partition, spawn decisions MUST remain locality-scoped and auditable at that finer-grained population boundary.
* Descendant spawn, if supported, MUST remain lineage-visible, locality-scoped, budget-visible, and membrane-mediated.
* A runtime MUST NOT place an actor or execution into a live running state unless spawn admission has already succeeded for that actor or execution.
* A runtime MUST NOT treat queued or budget-deferred spawn as equivalent to live running execution.
* In APSIX-governed and APSIX-replayable, MUST be membrane-mediated.

4.3. refine(zone_id, scope_ref, hypothesis) -> partition_set | error

Requests semantic decomposition or restructuring within a zone.

Requirements:

* MUST be recorded as a replayable event.
* MUST preserve lineage from prior scope.
* MUST NOT silently enlarge zone scope without explicit membrane approval.
* SHOULD preserve stable partition identity where possible.
* In APSIX-governed and APSIX-replayable, policy-sensitive refinement MUST be membrane-mediated.

4.4. execute(actor_id, target_ref, capability, input_refs) -> artifact | error

Applies a capability to a zone target or artifact target.

Requirements:

* MUST verify actor authorization for the requested capability.
* MUST verify target locality against admitted partitions or membrane-approved boundary crossing.
* MUST return an artifact or structured failure.
* MUST record provenance links to input references.
* MUST distinguish artifact-producing executions from effectful requests that attempt to mutate or publish across another authority boundary.
* An execution MAY produce artifacts representing further runtime requests such as refinement requests or spawn requests rather than final outputs.
* If the runtime materializes partition-local candidate outputs prior to harvest, it MUST keep those candidate outputs distinct from harvested authoritative outputs.
* If an execution is cancelled, aborted, or force-terminated by runtime policy, the runtime MUST expose a terminal outcome for that execution rather than leaving it in an ambiguous live state.
* Retries, recoveries, and resumed executions SHOULD be represented as distinct runs rather than silently overwriting prior execution outcomes.
* If the runtime supports external or cross-boundary effects, APSIX-governed and APSIX-replayable profiles MUST subject those effects to membrane mediation and auditable event records before completion.

4.5. anchor(artifact_ref, zone_id) -> anchor_id | error

Promotes an artifact into anchored state.

Requirements:

* MUST create or confirm stable artifact identity.
* MUST create an anchor record.
* MUST record decision provenance sufficient for replay.
* MUST NOT silently promote generated artifacts to anchored state without an explicit anchoring transition.
* In APSIX-governed and APSIX-replayable, MUST be membrane-mediated.

4.6. harvest(zone_id, filter) -> artifact_set

Returns anchored artifacts matching a filter or harvest profile.

Requirements:

* MUST NOT implicitly anchor unanchored artifacts.
* SHOULD support deterministic filtering.
* SHOULD preserve or validate the zone's authoritative output boundary; if that boundary has drifted since zone creation, a governed runtime SHOULD deny or escalate harvest rather than silently reinterpreting outputs.
* If the runtime uses partition-local candidate surfaces, harvest MUST perform an explicit transition into the authoritative output boundary rather than treating candidate-local paths as already harvested final outputs.
* MUST emit a harvest event.

4.7. freeze(zone_id) -> lifecycle_state | error

Transitions a zone into a closure regime.

Requirements:

* MUST deny further expansion primitives by default after success.
* MUST emit a freeze or closure event.
* SHOULD clean zone-owned ephemeral runtime resources such as worktrees, temporary branches, sandboxes, or equivalent execution residues.
* MUST NOT be required to re-harvest already harvested outputs in order to stabilize the zone.
* MUST NOT make previously anchored artifacts unreachable for replay or later harvest unless an explicit abandonment, archival, or retention policy is itself recorded as authoritative state.
* SHOULD preserve harvest and replay access.

4.8. observe(zone_id) -> event_stream | event_slice

Returns runtime events for the zone.

Requirements:

* MUST expose membrane decisions and artifact lifecycle transitions.
* MUST expose external effect decisions and outcomes, if such effects are supported.
* MUST expose terminal failure, cancellation, or abort outcomes for actor executions when the runtime supports those outcomes.
* MUST preserve authoritative ordering semantics.

* In APSIX-governed and APSIX-replayable, ordering SHOULD be reconstructible through a stable cross-ledger ordering surface such as seq_no, not only through wall-clock timestamps.
* In APSIX-replayable, request/decision/run correlation surfaces SHOULD be observable.
* Decision and event surfaces MUST be sound with respect to authoritative zone state: an implementation MUST NOT emit or retain admission, anchor, or lifecycle records that assert a zone state transition that did not occur in authoritative state.
* MAY paginate or stream.

5. Capability and Locality Model

Capabilities are primitive permissions in APSIX. Roles are derived from them.

5.1. Core Capability Set

An implementation SHOULD support capability names equivalent to:

* refine
* spawn
* execute
* anchor
* harvest

Equivalent implementation-local names MAY be used if their semantics are documented. Implementations MAY define richer capability vocabularies above this primitive substrate.

freeze() is a zone-administrative closure primitive and need not be exposed as a general actor capability.

5.2. Capability Mask

Every actor MUST operate under a capability mask. The runtime MUST reject attempts to execute a primitive that exceeds the actor’s mask.

5.3. Locality Binding

Capability grants MUST be interpreted relative to locality. Authorization is incomplete unless the runtime can answer both:

* which operation class is permitted;
* over which admitted partition set it is permitted.

5.4. Role Derivation

Roles are not normative objects in APSIX. A runtime MAY expose roles for ergonomics, but the normative substrate MUST remain the capability mask plus locality binding.

6. Membrane Semantics

The membrane is the critical enforcement locus of APSIX.

6.1. Complete Mediation

The runtime MUST subject the following operations to membrane mediation:

* actor creation
* policy-sensitive refinement
* cross-partition effects
* cross-zone effects
* external effects against other authority boundaries
* artifact promotion
* budget-sensitive expansion

If an implementation allows any of these operations to bypass membrane checks, it is not compliant with governed profiles.

6.2. Decision Surface

Membrane decisions MUST expose at least three outcomes:

* allow
* deny
* escalate

escalate indicates that local runtime authority is insufficient and an external approval or higher-level policy decision is required.

6.3. Decision Record

Each membrane-mediated decision MUST make available:

* decision
* request_id
* policy_version
* reason_code
* budget_context
* capability_basis

An implementation SHOULD expose this through a stable membrane_decision_record schema rather than through framework-local logging formats.

6.4. Budget Interaction

Budget exhaustion MUST be visible to membrane decisions. An exhausted zone MUST NOT continue infinite spawn or refinement as if the runtime had no control surface. If an implementation supports effect quotas or publication limits, exhaustion of those limits MUST likewise be visible to membrane decisions.

If an implementation supports multiple actors within one partition, budget and admission state SHOULD remain interpretable at that partition-local expansion boundary rather than only at whole-zone granularity.

Deferral due to budget or capacity pressure MUST remain distinguishable from execution failure. A runtime MUST NOT collapse budget_exhausted, queued, or deferred conditions into terminal actor failure unless an execution actually ran and failed.

7. Artifact, Provenance, and Replay Semantics

APSIX is not only about actor lifecycle. It is also about stable artifact handling and authoritative replay.

7.1. Artifact Lifecycle

An implementation MUST support a lifecycle in which artifacts are at minimum:

1. generated
2. optionally anchored
3. optionally harvested

If an implementation distinguishes between partition-local candidate outputs and authoritative final outputs, that distinction SHOULD remain explicit in artifact type, lifecycle state, or equivalent durable metadata.

7.2. Provenance

Anchored artifacts MUST preserve enough provenance to answer:

* which actor produced this artifact;
* in which zone;
* over which partition scope;

* under which capability;
* from which input references;
* under which membrane decision context.

At minimum, an anchored artifact SHOULD be portable together with:

* its anchor_record;
* the originating membrane_decision_record;
* referenced input artifacts or scope identifiers.

If an anchored artifact is the basis for an external effect, the runtime SHOULD preserve a portable reference to the effect decision and effect outcome record as well.

7.3. Replay Contract

An implementation MUST support authoritative reconstruction of zone state from:

* ordered events;
* membrane decision records;
* anchored artifacts;
* run records, if runs are exposed separately from actors;
* policy version history;
* budget and lifecycle checkpoints, if such checkpoints are used.

Replay MUST reconstruct authoritative runtime state. It MUST NOT be interpreted as a requirement to reproduce original latent reasoning traces.

For portability, replay-relevant records MUST be serializable through stable schemas rather than through implementation-specific debug logs alone.

If replay-relevant data is split across multiple ledgers, replay MUST reconstruct a single authoritative order across them.

When actor executions are cancelled, aborted, or otherwise force-terminated, replay MUST reconstruct those executions as terminal outcomes rather than as still-live runtime activity.

When execution is interrupted by runtime restart, process disappearance, or other loss of live actor state, a governed or replayable runtime SHOULD normalize the affected execution into a terminal, recovered, or requeued state rather than leaving authoritative state ambiguous.

If an implementation supports within-partition actor populations, replay SHOULD also reconstruct the effective spawn regime and lineage context sufficiently to distinguish bounded exploration from uncontrolled epidemic expansion.

If an implementation exposes operator-facing projections or planning views in addition to authoritative zone state, replay MUST treat those views as derived artifacts rather than as independent authoritative inputs.

7.4. No Silent Promotion

An implementation MUST NOT treat generated output as anchored by default unless generation and anchoring are explicitly collapsed into one documented primitive.

7.5. Measurement Convention

If an implementation exposes runtime telemetry, proxy metrics, or derived analytical surfaces in addition to the core APSIX objects, it SHOULD expose a stable measurement convention describing:

* which records or state surfaces are treated as measurement inputs;
* whether metrics are direct runtime variables or deployment-specific proxies;
* what aggregation horizon or windowing assumptions apply;
* what runtime profile or workload regime the measurements are valid for.

Implementations MAY expose population metrics describing actor population size, spawn rate, partition-local population distribution, or other indicators useful for detecting uncontrolled expansion regimes.

If such metrics are used to support replayable analysis, comparison across zones, or theory-mapping claims, APSIX-replayable implementations SHOULD make that measurement convention visible through a stable manifest, report schema, or equivalent replay-relevant surface.

8. Error Model

Primitive failures SHOULD be structured rather than opaque.

Recommended error classes:

* policy_denied
* budget_exhausted

* unknown_zone
* unknown_actor
* unknown_artifact
* unknown_partition
* capability_denied
* invalid_transition
* effect_denied
* requires_escalation
* replay_unavailable
* aborted
* cancelled

When observe(zone_id) is implemented, the runtime SHOULD make these errors observable through it. APSIX-governed and APSIX-replayable profiles MUST do so.

9. Compliance Profiles

This document defines three progressive profiles.

9.1. APSIX-core

Requires:

* zone
* refine
* spawn
* execute
* anchor
* harvest
* freeze

This profile is the minimal portable primitive surface.

9.2. APSIX-governed

Requires all APSIX-core primitives plus:

* observe
* membrane-mediated spawn
* membrane-mediated refine for policy-sensitive scope changes
* membrane-mediated external or cross-boundary effects, if the runtime supports them
* membrane-mediated anchor
* explicit budget handling
* decision records with reason codes
* stable membrane decision records

* stable request correlation across membrane-mediated lifecycle transitions

This profile is the minimum useful profile for real spawn governance.

9.3. APSIX-replayable

Requires all APSIX-governed features plus:

* partition-visible locality semantics
* artifact provenance
* explicit run records or equivalent execution-attempt visibility
* stable anchor records
* ordered event exposure
* stable event records
* stable cross-ledger ordering semantics
* ledger manifest or equivalent replay surface description
* policy version visibility
* authoritative replay support
* measurement convention visibility, if runtime telemetry proxies are exposed

This profile is the recommended basis for research and auditable production systems.

9.4. Conformance Guidance

Primitive presence alone is insufficient to establish meaningful APSIX conformance.

Recommended conformance evaluation SHOULD additionally check:

* lifecycle ordering invariants across zone, spawn, execute, anchor, harvest, and freeze;
* request/decision/run correlation integrity where those surfaces are exposed;
* preservation of authoritative cross-ledger ordering semantics;
* absence of silent promotion from generated to anchored state;
* explicit separation between candidate-local artifact surfaces and harvested authoritative output surfaces, when both exist;
* distinction between deferred/queued conditions and terminal execution failure;
* normalization of interrupted executions into terminal, recovered, or requeued authoritative states;
* preservation of locality-scoped spawn control when multiple actors per partition are supported;
* consistency between authoritative zone state and exposed event or decision ledgers.

Implementations MAY satisfy these checks through scenario traces, replay fixtures, or other stable lifecycle test surfaces rather than only through isolated primitive-level tests.

10. Mapping to Concrete Runtimes

APSIX does not mandate a specific implementation style.

Examples:

* a content-spawn runtime could treat each source idea as one zone and each channel draft as a harvestable artifact family;
* a software runtime could treat bounded codebase areas as zones and patches or test reports as artifacts;
* a research runtime could treat a literature field as a zone and claim-evidence memos as anchored artifacts.

What matters is not the domain itself, but whether the runtime exposes the primitives, locality semantics, membrane decisions, and replay obligations coherently.

11. Non-Goals

APSIX does not attempt to standardize:

* chain-of-thought formats;
* inter-model competition protocols;
* persona taxonomies;
* benchmark task schemas;
* human-interface conventions;
* internal scheduler algorithms;
* concrete effect transports or publication APIs;
* storage and graph-engine implementations.

Those may exist above or below the APSIX layer, but they are not part of the core contract.

12. Open Issues

Several questions remain open for later versions:

* whether cross-zone merge() belongs in an extension set;
* whether an explicit effect() primitive should remain derived from execute() plus membrane policy or become standardized;
* what minimum provenance fields are sufficient for strong auditability;
* whether event, anchor, and membrane decision schemas should be versioned independently or under one runtime schema version;

* how replay semantics should interact with distributed asynchronous runtimes;
* whether an explicit checkpoint() primitive should remain derived or become standardized;
* whether Run should remain a logical object or become a first-class primitive argument surface;
* whether output-boundary drift detection should become mandatory in governed profiles or remain a replayable-profile recommendation.

13. Security Considerations

APSIX defines a governance surface for spawn-capable semantic runtimes. As a result, security and governance failures in an APSIX implementation are not limited to data exposure; they can also alter population growth, artifact promotion, and cross-boundary effects.

Improper membrane policy configuration, incomplete mediation, or weak audit surfaces may lead to:

* uncontrolled actor population expansion;
* artifact promotion of unverified or policy-incompatible outputs;
* cross-boundary effect escalation without sufficient auditability;
* replay gaps that prevent authoritative reconstruction of sensitive decisions;
* policy drift between authoritative zone state and exposed event or decision records.

Implementations SHOULD ensure that membrane policies enforce:

* strict admission control for actor creation and policy-sensitive refinement;
* explicit artifact validation prior to anchoring;
* auditable decision records for cross-boundary effects;
* protection of authoritative ordering and replay-relevant ledgers against silent tampering or divergence;
* clear separation between generated, anchored, and harvested artifact states.

Implementations supporting external effects SHOULD additionally ensure that:

* escalation paths are authenticated and authorization boundaries are explicit;
* effect completion is recorded as a durable auditable outcome;
* cancelled, aborted, or partially completed effects are not presented as successful state transitions.

Failure to enforce these constraints may result in loss of governance over semantic expansion within a zone and may permit unauthorized external actions to appear compliant in operator-facing views.

14. IANA Considerations

This document has no IANA actions.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

16. Conclusion

APSIX is a proposal for a minimal primitive contract for portable semantic runtimes. Its central claim is that the portable layer should standardize zone, membrane, run, artifact, locality, and replay operations rather than framework-local agent behaviors.

If this direction is useful, higher-level runtimes and applications can target a stable substrate of zones, spawn, refinement, execution runs, anchoring, harvest, membrane decisions, and replayable authoritative state. That would move the field one step away from ad hoc orchestration and one step closer to real runtime portability.