

TMPA Core Specification

Zhu Wei - joinwell52 Research Center

2026-08-11 - S1.0 - TMPA V1.0

Contents

TMPA Core Specification	2
Textual Multi-Agent Process Architecture — Core Objects, Reader Semantics, and Conformance . . .	2
1. Status, Scope, and Conformance Boundary	3
2. Terminology and Representation Stages	4
3. Core Architecture Model	5
3.1 Governance Object	5
3.2 Document-Type Registries	6
3.3 Role and Authority Model	6
3.4 Lifecycle Model	7
3.5 Textual Messages, Single-Writer Streams, and Asynchronous Parallelism	7
3.6 Read-Side Aggregation and Governance Reconstruction	8
3.7 Integrity and Signature Evidence	9
3.8 Governance Closure Abstracted from FCoP Practice	10
3.8.1 Current State, Transition History, and Business Completion	10
3.8.2 Reciprocity, Claims, and Evidence Gates	10
3.8.3 Parent-Child Work and Closure Roll-up	11
3.8.4 Governance Decisions, Risk, and Human Approval	11
3.8.5 Failure, Recovery, Inspection, and Drift	11
4. Canonical Object, Encoding, and Reconstruction	11
4.1 Canonical Object Schema	11
4.2 Canonical Textual Encoding Profile	14
4.3 Aggregation and Governance-Reconstruction Procedure	15
4.4 Conflict and Validation Handling	16
5. Threat Model and Trust Assumptions	16
5.1 Protected Properties	16
5.2 Trust Roots	16
5.3 Threats Considered	17
5.4 Malicious Participants and Storage-Surface Compromise	17
False-but-well-formed evidence at publication time	17
5.5 Levels of Role and Identity Separation	18
5.6 AI Agent Identity	19
5.7 Security Claims	19
6. Lifecycle and Authority Evaluation	19
6.1 Required Registries	19
6.2 Transition Evaluation Order	20
6.3 State Reconstruction	20

6.4 Authority Time and Revocation	20
7. Three-Valued Governance Logic	20
7.1 Judgment Domain	20
7.2 Primitive Classification Rules	20
7.3 Composition Rules	21
7.4 Judgment and View Mapping	21
8. Reader Input and Output Contract	21
8.1 Input Bundle	21
8.2 Canonical Result	22
8.3 Canonicalization and Ordering	22
8.4 Failure and Partial Output	23
9. Normative TMPA Core Specification	23
9.1 Normative Language	23
9.2 Object Requirements	23
9.3 Type Registry Requirements	23
9.4 Role Requirements	24
9.5 Stream Requirements	24
9.6 Lifecycle Requirements	24
9.7 Reference Requirements	25
9.8 Integrity Requirements	25
9.9 Aggregation and Reader-Reconstruction Requirements	26
9.10 Recovery Requirements	26
9.11 Governance Closure, Claims, and Human-Control Requirements	27
10. Conformance and Testability	27
10.1 Conformance Levels	27
10.2 Required Conformance Tests	28
10.3 Executable Test-Case Contract	29
10.4 Verdict Algorithm and Conformance Claim	29
10.5 Test Fixtures and Result Reporting	30
10.6 Compliance Crosswalks	30
11. Profile, Versioning, and Publication Rules	31
11.1 Core and Profile Separation	31
11.2 Versioning	31
11.3 Publication and Evidence Boundary	31
11.4 S1.0 Release and Evidence Record	31
11.5 S0.5 FCoP-Derived Historical Baseline	32
Appendix A. Historical Source Traceability (Informative)	32
Appendix B. FCoP Source Crosswalk (Informative)	32

TMPA Core Specification

Textual Multi-Agent Process Architecture — Core Objects, Reader Semantics, and Conformance

Specification Version: Stable Release S1.0

Historical Extraction Baseline: TMPA Draft V1.0-R24; current specification maintained directly in this GitHub document

Status: Official stable publication

Extraction Date: 2026-07-31

Release Date: 2026-08-11

Authority: This GitHub document is the sole normative source for TMPA Core S1.0. The Architecture Paper is theoretical and the Implementation Case Report is evidentiary; neither may redefine this specification.

1. Status, Scope, and Conformance Boundary

This document defines the vendor-neutral TMPA Core. It fixes the canonical governance object model, single-writer and lifecycle semantics, source aggregation, deterministic governance reconstruction, three-valued judgment algebra, reader output contract, trust assumptions, normative requirements, and C01–C14 conformance behavior.

TMPA Core governs process-responsibility evidence. It does not prescribe a model runtime, scheduler, storage engine, message transport, database, filesystem layout, identity provider, key-management system, or enterprise control plane. A profile may bind those mechanisms, but it **MUST** preserve the Core semantics and **MUST** identify every additional assumption or guarantee.

This stable release separates three kinds of text:

- **normative requirements:** Sections 9 and 10, using the terms defined in Section 9.1;
- **architectural and algorithmic definitions required to interpret the normative clauses:** Sections 2–8;
- **informative extraction and traceability notes:** Section 11 and Appendix A.

The following are outside TMPA Core unless a named profile adds them: authenticated enterprise identity, credential issuance, recursive delegation, runtime admission control, protected or tamper-resistant storage, distributed consensus, semantic truth verification, Byzantine resilience, legal certification, and jurisdiction-specific compliance.

A conforming implementation **MAY** use files, database rows, object-store objects, or events as source artifacts. It **MUST NOT** claim conformance merely because it stores Markdown, produces logs, or implements a workflow state machine. Conformance is behavioral and depends on the observable object, lifecycle, reconstruction, conflict, recovery, and test requirements in Sections 9 and 10.

The publication contract is fixed across the three maintained documents: the Architecture Paper A1.0 explains the theory, this Core Specification defines normative behavior, and the Implementation Case Report I1.0 reports bounded engineering evidence. Their current guidance relation and historical co-evolution are distinct:

CURRENT GUIDANCE AND IMPLEMENTATION

TMPA theory and architecture
↓ formalized as normative behavior by
TMPA Core Specification
↓ projected through the file-based profile
FCoP protocol
↓ used by
CodeFlowMu engineering system

HISTORICAL CO-EVOLUTION

XiaoDian AI practice → early TMPA → FCoP extraction and maturation
→ CodeFlowMu engineering implementation
FCoP + CodeFlowMu feedback → current TMPA formalization

TMPA theory guides CodeFlowMu engineering, this Core fixes the normative behavior under evaluation, and FCoP provides the coordination protocol used by CodeFlowMu. The `fcop` and `fcop-mcp` Python packages are reference implementations of the protocol, not FCoP itself. Engineering feedback may motivate a later TMPA revision, but neither the FCoP protocol, its reference implementation, nor CodeFlowMu defines or exhausts TMPA Core.

2. Terminology and Representation Stages

The paper fixes the following vocabulary so that semantic objects, physical storage, message behavior, and reconstructed views are not treated as interchangeable concepts.

Canonical English term	Fixed Chinese equivalent	Fixed meaning	Not equivalent to
governed work item	受治理工作项	the task, request, decision, or process subject whose responsibility and lifecycle are being governed	one file, one session, or one runtime job
primary carrier	主载体	the stable governance object that anchors the identifier and minimum governing context of one work item	a mutable record that every participant edits
governance object	治理对象	one canonical semantic unit authored by one creator under one responsible role and one writer stream	its storage path, transport envelope, or derived view
textual message	文本消息	the communication function of a governance object when it transfers work, evidence, review, or decision semantics	a separate object class or an ephemeral queue message
state carrier	状态载体	the persistence function through which an object, transition record, or profile-defined location contributes declared or current state evidence	shared mutable application state
source artifact	来源工件	one physical representation or observation of evidence, such as a file, database row, object-store item, or received event	the semantic governance object after validation
source candidate	来源候选	one discovered source artifact presented to the aggregation stage, including malformed or conflicting observations	an accepted authoritative object
canonical candidate set	规范候选集合	the source-preserving, parsed, indexed, and deterministically normalized collection returned by aggregation	the final governance conclusion
writer stream	写者流	the locally ordered sequence of governance objects published by one attributable writer	a global event log or total timeline
source aggregator	来源聚合器	the stage that discovers, preserves, parses, indexes, and normalizes source candidates without deciding governance truth	the governance reader
governance reader	治理 Reader	the deterministic stage that applies a fixed profile to the canonical candidate set	the storage layer, orchestrator, or model runtime
governance graph and issue set	治理图与问题集合	the reconstructed partial-order process view and the canonical unresolved-condition output	the original source evidence or an imposed total order

Canonical English term	Fixed Chinese equivalent	Fixed meaning	Not equivalent to
lifecycle state	生命周期状态	the current profile stage reconstructed from valid transitions or state observations	business acceptance, semantic truth, or proof that intended value was delivered
business acceptance	业务验收	an independent conclusion by an authorized role over a delivery claim and its supporting evidence	executor self-report, terminal storage location, or a done label
completion claim	完成声明	an inspectable assertion that a work item, deliverable, or child-work set satisfies its requirements	a self-authenticating completion fact
work derivation	工作派生关系	an explicit parent-child relation retaining scope, responsibility, and closure requirements	a transient runtime fork or an unproven task list
governance decision	治理裁决	an independent review, approval, rejection, change request, abstention, or human-escalation object	the lifecycle review stage itself
inspection finding	巡检发现	a reproducible drift or risk signal emitted by a reader, auditor, or governance-alert component	an automatically executed repair or business decision

The three semantic judgments are also fixed across all documents: **valid** (有效) means the applicable evidence and rules establish acceptance; **invalid** (无效) means the rules establish rejection or violation; **undetermined** (未确定) means evidence is incomplete, conflicting, or awaiting an authorized resolution. View labels such as authoritative, quarantined, partial, disputed, and pending_human explain the presentation reason; they are not additional semantic values.

A single canonical governance object may be realized by different physical profiles. In FCoP, its source artifact is ordinarily a file plus path and event evidence; another profile may use a row, object, or event. Conversely, two source artifacts that declare the same object identifier but contain different canonical content are not two harmless copies: they are conflicting candidates that must be retained and evaluated under the profile. Throughout the architecture and normative chapters, **object** refers to the semantic unit, **artifact** to a physical or published engineering representation, and **view** to a reader-derived result.

3. Core Architecture Model

TMPA defines governance semantics, not a runtime component. Its architecture specifies which governance facts must be represented, how responsibility and lifecycle are expressed, and how independent evidence is reconstructed into an authoritative view. Storage, transport, scheduling, and model behavior remain implementation concerns unless a TMPA profile explicitly binds them.

3.1 Governance Object

A **governance object** is the smallest independently attributable semantic unit in TMPA. It may represent a task, report, review, approval, issue, lifecycle transition, role assignment, recovery action, or another document type published by a protocol profile. The object is distinct from its physical **source artifact**: FCoP may encode it as a file and path observation, while another profile may use a database row, object-store item, or event.

The terms **textual message** and **state carrier** describe functions of an object rather than additional object classes. An object acts as a textual message when it transfers governed work or evidence, and it acts as a state carrier when its content, transition evidence, or profile-defined location contributes to lifecycle reconstruction.

Every governance object contains or identifies:

- a stable object identifier;
- a document type;
- one creator identity;
- one responsible role;
- a stream identifier and sequence number;
- a creation time;
- a lifecycle profile and declared state;
- typed references to related objects;
- optional parent-work and thread identifiers;
- optional completion, failure, recovery, or acceptance claims and their evidence references;
- a risk level and human-approval requirement when required by the profile;
- canonical textual content;
- integrity evidence.

A published governance object is immutable. Correction does not erase or rewrite the original object; it creates a new object that supersedes, rejects, qualifies, or resolves the earlier one. Multiple byte-identical source observations may refer to the same object without changing its meaning; the same identifier paired with different canonical content is a conflict, not an update.

The lifecycle state declared by an object is the state associated with that object under its profile at publication. The current authoritative state of governed work is reconstructed from the valid object set, accepted transitions, and profile rules. It is not obtained by mutating an earlier published object or by selecting the most recent timestamp.

Lifecycle state and business acceptance are orthogonal dimensions. A terminal state, archive location, or done declaration is state evidence only. It **MUST NOT** be reconstructed as business completion unless the applicable profile defines acceptance authority, required delivery evidence, and separation of duties, and the reader finds a valid acceptance object.

3.2 Document-Type Registries

TMPA Core does not impose a universal business-document taxonomy. Each implementation profile instead publishes a finite document-type registry.

Each registry entry defines:

- the type name and version;
- the governance responsibility represented by the type;
- permitted creator roles;
- required fields;
- permitted reference relations;
- the applicable lifecycle profile;
- validation rules.

Document types must not overlap ambiguously in authority. An execution report, for example, does not implicitly serve as its own independent review or approval. When a deployment permits an exception to separation of duties, the exception and its approving authority must be represented explicitly.

3.3 Role and Authority Model

A TMPA role is a governance authority with a defined scope. It is not merely a prompt label or natural-language persona.

Each role definition identifies:

- a stable role identifier;
- permitted object types;
- permitted lifecycle actions;
- separation-of-duty constraints;
- the authority that assigns the role;
- the assignment's validity period and revocation state.

A profile MAY organize roles into execution, governance, and administration capability layers, but it must define permitted and prohibited directions and identify the identity or runtime control that enforces them. Declared capability boundaries and enforced capability boundaries MUST be reported separately.

An object's role field declares the authority under which the creator acted; it does not create that authority. A reader validates the claim against an active role assignment and the applicable policy profile.

A participant may occupy more than one role only when the implementation profile explicitly permits the combination. A deployment claiming independent review must prohibit the same identity from acting as both executor and reviewer for the same governed result unless a recorded and authorized exception applies.

In an enterprise identity profile, the logical role is bound separately to a verifiable agent or workload identity, the human or organizational principal that remains accountable, the credential used for the action, the delegation source and scope, and the validity or revocation state. TMPA Core records and validates the governance claim; it does not issue credentials or enforce recursive permission attenuation.

3.4 Lifecycle Model

A lifecycle profile consists of:

- a finite state set S ;
- an initial state s_0 ;
- a terminal-state set F ;
- an action set A ;
- a transition relation $T \subseteq S \times A \times S$;
- an authorization function $\text{Auth}(\text{role}, \text{action}, \text{object})$;
- a validation function $\text{Valid}(\text{object}, \text{transition})$.

A transition is accepted only when:

1. its source state matches the current authoritative state;
2. the transition is defined by T ;
3. the initiating role is authorized;
4. required references and preconditions are satisfied;
5. the transition evidence passes schema and integrity validation.

An illegal or unauthorized transition does not alter the authoritative lifecycle state. The attempt remains observable through a rejection, issue, alert, or equivalent profile-defined record rather than being silently discarded or repaired.

A lifecycle profile MUST additionally define: (1) which states require independent business acceptance; (2) which relations constitute reporting, review, acceptance, and archive authorization; (3) how parent and child work roll up; (4) which risk levels require human approval; and (5) failure types, recovery actions, and their persistent evidence. Lifecycle tools or physical locations may implement these rules but do not replace their semantic definitions.

3.5 Textual Messages, Single-Writer Streams, and Asynchronous Parallelism

TMPA's write plane combines a stable work carrier, single-writer objects, local seriality, and asynchronous composition.

For every governed task or work item t , a task-oriented profile defines one stable primary carrier c_t . The carrier establishes the identifier and minimum governing context of the work. Acceptance, execution reports, reviews, decisions, corrections, and recovery records are separate objects that reference c_t ; they do not become additional mutable copies of the task. “One task, one carrier” therefore means one stable primary reference point, not one document that every participant edits.

Let A be the set of responsible writers. Every published object has exactly one writer, and each writer $a \in A$ publishes an independently attributable serial stream:

$$S_a = \langle o_{\{a,1\}}, o_{\{a,2\}}, \dots, o_{\{a,n\}} \rangle$$

The sequence inside S_a is authoritative local order. Every object has a positive sequence number, and $(stream_id, sequence)$ identifies its position within that writer's responsibility history. Creation time is informative but not authoritative for stream order.

At observation time τ , the available candidate collection may contain different prefixes of different streams:

$$O_\tau = \text{union}_{\{a \in A\}} \text{prefix}(S_a, k_a(\tau))$$

The functions $k_a(\tau)$ need not advance together. One participant may publish a task while another is offline; a report may appear before an independent review; several writers may progress concurrently. This is how multiple serial streams form asynchronous parallelism. TMPA does not require all participants to share a clock, remain online together, or commit to one global event log.

Within-stream predecessor relations provide local order. Explicit references and profile-defined lifecycle dependencies provide cross-stream causal edges. If two objects have neither a within-stream relation nor a profile-defined cross-stream dependency, they remain concurrent and incomparable.

Single-writer objects and separate responsibility streams remove the primary **semantic** shared-write conflict: several agents do not compete to edit one authoritative record. They do not eliminate every storage-level contention, filesystem race, or infrastructure failure; profiles must still define atomic publication, duplicate handling, and recovery behavior.

The write model can be summarized as:

One task has one primary carrier. One writer owns each published object. Each writer remains serial. Multiple streams progress asynchronously to form parallel collaboration.

3.6 Read-Side Aggregation and Governance Reconstruction

TMPA's read plane has two conceptually separate stages: **source aggregation** and **governance reconstruction**.

Let O_τ be the finite collection of source candidates visible at observation time τ . A source-preserving aggregator A discovers source artifacts, retains source identity and bytes, parses candidate envelopes, indexes identifiers and references, and applies deterministic normalization needed by the reader:

$$C_\tau = A(O_\tau)$$

Aggregation does not decide which claim is true, silently repair a conflict, invent missing evidence, or convert arrival order into governance order. Its purpose is to construct the complete canonical candidate set C_τ available to the governance reader while retaining the provenance of every source candidate.

Let P be a fixed rule profile containing schemas, type rules, role assignments, lifecycle rules, relation semantics, canonicalization rules, conflict policy, and output-normalization rules. The governance reader then computes:

$$R_P(C_\tau) = (G_\tau, I_\tau)$$

where G_τ is the canonical reconstructed partial-order process and governance graph and I_τ is the canonical issue set. G_τ represents workflow progress, responsibility, lifecycle, review, approval, rejection, recovery, and audit relations while preserving local stream order, explicit cross-stream dependencies, and concurrency among incomparable objects. It is not an authoritative total timeline.

For brevity, later sections may write $R_P(0)$ for the composed pipeline $R_P(A(0))$. The primary determinism requirement is permutation invariance. For every permutation π of the same canonical candidate collection:

$$R_P(A(\pi(0))) = R_P(A(0))$$

The equality applies to canonical serialization of both G and I , not to incidental in-memory order or diagnostic formatting. Delayed arrival changes the currently available set and may legitimately change a view from partial to authoritative or disputed; different enumeration orders of the same set must not change the result.

Determinism proposition. Let 0 be a finite source multiset and P a fixed profile. Assume that: (1) source normalization is a pure function of source identity and covered bytes; (2) duplicate classification, validation, authority checks, lifecycle checks, and issue identifiers are functions of canonical object values and P ; (3) graph edges are derived only from within-stream sequence and profile-declared relations; and (4) graph and issue serialization use published deterministic ordering and tie-break rules. Under these conditions, the composed operation $R_P(A(0))$ is invariant to source enumeration order.

Proof sketch. Aggregation maps any enumeration of 0 to the same canonical indexed candidate multiset because indexing and duplicate classification depend on canonical source values rather than discovery position. Per-object validation is order-independent. Set-level checks—duplicate identifiers, stream gaps, missing references, prohibited cycles, authority conflicts, and lifecycle conflicts—are computed over the same canonical sets and relations. The accepted node set and directed edge set are therefore identical for every permutation. Canonical issue identifiers and deterministic ordering produce the same I ; canonical graph serialization and a stable tie-break used only for representing incomparable nodes produce the same serialized G . Consequently, every permutation yields byte-equivalent canonical outputs. This argument establishes permutation invariance under the stated profile contract; it does not prove semantic truth, profile correctness, resistance to compromised trust roots, or equality across different evidence sets. A mechanized proof and executable corpus remain required for stronger assurance.

A reconstructed subject or subgraph is classified as:

- **authoritative** when required evidence is valid and no unresolved integrity, authority, lifecycle, ordering, or required-reference issue affects the conclusion;
- **partial** when required evidence is missing or a stream or dependency is incomplete;
- **disputed** when multiple valid but incompatible governance claims remain unresolved;
- **quarantined** when a profile-defined integrity, identity, duplicate-ID, or prohibited-cycle condition excludes the affected evidence from authoritative reconstruction.

Authentication is an orthogonal assurance label. An object or view may be structurally authoritative under TMPA Core while remaining unauthenticated under a stronger identity profile; it must not then be presented as authenticated integrity.

TMPA requires **conflict preservation**: valid contradictory objects remain represented until a new authorized resolution object exists. It also requires **evidence preservation under extension**: adding candidate evidence does not erase prior source evidence. TMPA does not assume monotonicity of governance status under arbitrary set extension, because newly added valid evidence may legitimately change a previously authoritative view into a partial, disputed, or quarantined one.

Read-side reconstruction is therefore not the whole architecture; it is the stage that converts durable textual messages and asynchronous serial streams into a coherent process and governance result. A deterministic topological serialization may be generated for interchange or display, but that serialization does not add governance order between incomparable nodes. C11 operationalizes source-aggregation and reconstruction determinism; C03, C04, C09, C10, and C12 exercise identity, ordering, dependency, cycle, and conflict-preservation behavior. A mechanized proof of the full reconstruction properties remains future work.

3.7 Integrity and Signature Evidence

TMPA separates three properties that are often conflated:

1. **Attribution:** an object declares a creator and responsible role.
2. **Integrity:** modification of a published object can be detected.
3. **Authenticated integrity:** the object is cryptographically bound to a verified identity or key.

TMPA Core requires attribution and integrity evidence. A deployment may claim authenticated integrity only when it applies a trusted identity, signature, and key-management profile.

An integrity record identifies:

- the canonicalization profile;
- the hash algorithm;
- the content digest;
- predecessor or referenced digests when required by the profile;
- an optional signature algorithm;
- an optional key identifier;
- an optional signature value.

A role label is not a cryptographic signature. A digest without a trusted identity binding can detect modification but cannot prove who created the object. A valid signature proves origin and integrity under the deployed trust model; it does not prove that the signed content is semantically true.

3.8 Governance Closure Abstracted from FCoP Practice

S1.0 derives vendor-neutral Core constraints from FCoP protocol specifications, Rules, Schemas, and ADRs, together with bounded observations of the `fcop` / `fcop-mcp` reference implementation. FCoP is a protocol and reference profile, not an application and not the definition of TMPA; the Python packages are only its reference implementation. This section therefore absorbs portable semantics rather than `_lifecycle/`, `filename`, or `MCP-tool` names.

This historical extraction is implementation feedback into specification design, not a reversal of current authority. Once published, TMPA theory and this Core govern the intended CodeFlowMu implementation; observed product behavior can support, challenge, or motivate a later revision, but cannot silently redefine the current requirements.

3.8.1 Current State, Transition History, and Business Completion

A profile **MUST** define current-state observations separately from transition-history evidence. FCoP uses `path` as current stage and `append-only transitions` as history; a database profile may use a current-state row and event table. When they conflict, the reader **MUST** preserve both sources and emit `STATE_EVIDENCE_CONFLICT` or a profile-declared canonical equivalent. It **MUST NOT** resolve the conflict by selecting the latest timestamp.

Entering `done`, a terminal state, or an archive **MUST NOT** automatically establish business acceptance. The executor report is an attributable delivery claim; only a review or acceptance object issued by an authorized and sufficiently independent actor can establish business completion. Without that object, the completion conclusion is undetermined and the view is `partial` or `pending_human`.

3.8.2 Reciprocity, Claims, and Evidence Gates

A work-oriented profile **MUST** define reciprocal relations between work requests and responses. Every accepted work object **SHALL** eventually relate to a report, issue, rejection, cancellation, or follow-up work object; silence cannot be inferred as success.

Completion, failure, recovery, and acceptance assertions **MAY** be represented by `claims`. Every claim has a stable claim identifier, predicate, subject, and evidence-object identifier set. If a completion claim lacks tests, artifacts, commits, reports, or other evidence required by the profile, the reader **SHALL** emit `CLAIM_EVIDENCE_MISSING` and retain undetermined. This rule governs unsupported claims; it does not claim to eliminate model hallucinations.

3.8.3 Parent-Child Work and Closure Roll-up

Child work SHALL identify its parent through `governed_work.parent_id` or a profile-declared equivalent. A shared thread may be represented by `thread_id`, but a thread MUST NOT replace the parent-child scope relation. When a parent has an unfinished child, an unhandled blocked child, or a child without a reciprocal outcome, a parent completion claim SHALL emit `CHILD_WORK_OPEN` and remain undetermined.

Scope correction MUST be expressed by a new object, supersession relation, or new derivation relation rather than in-place rewriting of a published parent. The reader SHALL preserve the parent, every child, their reciprocal outcomes, and the roll-up conclusion.

3.8.4 Governance Decisions, Risk, and Human Approval

A lifecycle review stage and a governance-decision object are orthogonal mechanisms. A profile MUST assign them different type or relation semantics. An implementation MUST NOT infer independent review merely because work entered a review stage, and a governance review must not replace the execution report.

A profile MAY use the risk levels `low`, `medium`, `high`, and `irreversible`. If an object declares that human approval is required, or its risk level belongs to the profile's human-approval set, the reader SHALL emit `HUMAN_APPROVAL_REQUIRED`, judgment `undetermined`, and view `pending_human` until a valid human-approval object exists. An agent cannot satisfy the requirement by rewriting its own decision.

3.8.5 Failure, Recovery, Inspection, and Drift

A profile MUST publish finite failure-type and recovery-action registries and define how retry, resume, roll-back, abort, and escalation produce new objects. Failure MUST NOT be hidden by a success report; a recovery object MUST reference both the triggering failure and the recovered work.

Protocol inspection and governance alerts are observations, not automatic remediation. `INSPECTION`, `ALERT`, or equivalent objects MAY report blocking, normative, or hygiene findings, but suggested commands MUST NOT be interpreted by the reader as executed transitions. Independent governance signals and executor self-reports MUST be classified separately.

4. Canonical Object, Encoding, and Reconstruction

4.1 Canonical Object Schema

The following JSON Schema defines the TMPA Core S1.0 canonical object representation. It constrains the shape of one governance object. Cross-object properties—including identifier uniqueness, stream continuity, role authorization, lifecycle legality, reference resolution, and deterministic reconstruction—are evaluated by the applicable profile and reader rather than by this single-object schema.

Implementations may add profile-specific fields only under extensions. They must preserve the meaning of the core fields.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:tmpa:schema:governance-object:s1.0",
  "title": "TMPA Governance Object S1.0",
  "$comment": "Structural validation does not establish role authority, lifecycle legality, cross-object uniqueness,
  ↪ or integrity verification.",
  "type": "object",
  "additionalProperties": false,
  "required": [
    "tmpa_version",
    "id",
    "type",
    "governed_work",
    "stream",
    "creator",
    "role",
    "created_at",
  ],
}
```

```

    "lifecycle",
    "references",
    "content",
    "integrity"
  ],
  "properties": {
    "tmpa_version": { "const": "S1.0" },
    "id": { "type": "string", "minLength": 1 },
    "type": { "type": "string", "minLength": 1 },
    "governed_work": {
      "type": "object",
      "additionalProperties": false,
      "required": ["id", "primary_carrier_id"],
      "properties": {
        "id": { "type": "string", "minLength": 1 },
        "primary_carrier_id": { "type": "string", "minLength": 1 },
        "parent_id": { "type": "string", "minLength": 1 },
        "thread_id": { "type": "string", "minLength": 1 }
      }
    },
    "stream": {
      "type": "object",
      "additionalProperties": false,
      "required": ["id", "sequence"],
      "properties": {
        "id": { "type": "string", "minLength": 1 },
        "sequence": { "type": "integer", "minimum": 1 }
      }
    },
    "creator": { "type": "string", "minLength": 1 },
    "role": { "type": "string", "minLength": 1 },
    "created_at": { "type": "string", "format": "date-time" },
    "lifecycle": {
      "type": "object",
      "additionalProperties": false,
      "required": ["profile", "state"],
      "properties": {
        "profile": { "type": "string", "minLength": 1 },
        "state": { "type": "string", "minLength": 1 },
        "transition": {
          "type": "object",
          "additionalProperties": false,
          "required": ["from", "action", "to"],
          "properties": {
            "from": { "type": "string", "minLength": 1 },
            "action": { "type": "string", "minLength": 1 },
            "to": { "type": "string", "minLength": 1 }
          }
        }
      }
    },
    "references": {
      "type": "array",
      "uniqueItems": true,
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["relation", "target"],
        "properties": {
          "relation": { "type": "string", "minLength": 1 },
          "target": { "type": "string", "minLength": 1 }
        }
      }
    },
    "claims": {
      "type": "array",
      "uniqueItems": true,
      "items": {
        "type": "object",
        "additionalProperties": false,
        "required": ["id", "predicate", "subject", "evidence"],
        "properties": {
          "id": { "type": "string", "minLength": 1 },
          "predicate": { "type": "string", "minLength": 1 },
          "subject": { "type": "string", "minLength": 1 },
          "evidence": { "type": "array", "uniqueItems": true, "items": { "type": "string", "minLength": 1 } }
        }
      }
    },
    "risk": {
      "type": "object",
      "additionalProperties": false,
      "required": ["level", "requires_human_approval"],

```

```

    "properties": {
      "level": { "enum": ["low", "medium", "high", "irreversible"] },
      "requires_human_approval": { "type": "boolean" }
    },
  },
  "content": {
    "type": "object",
    "required": ["media_type", "body"],
    "properties": {
      "media_type": { "type": "string", "minLength": 1 },
      "body": {}
    },
    "additionalProperties": false
  },
  "integrity": {
    "type": "object",
    "additionalProperties": false,
    "required": ["canonicalization", "hash_algorithm", "digest"],
    "dependentRequired": {
      "signature_algorithm": ["key_id", "signature"],
      "key_id": ["signature_algorithm", "signature"],
      "signature": ["signature_algorithm", "key_id"]
    },
    "properties": {
      "canonicalization": { "type": "string", "minLength": 1 },
      "hash_algorithm": { "type": "string", "minLength": 1 },
      "digest": { "type": "string", "minLength": 1 },
      "signature_algorithm": { "type": ["string", "null"] },
      "key_id": { "type": ["string", "null"] },
      "signature": { "type": ["string", "null"] }
    },
    "oneOf": [
      {
        "properties": {
          "signature_algorithm": { "type": "null" },
          "key_id": { "type": "null" },
          "signature": { "type": "null" }
        }
      },
      {
        "required": ["signature_algorithm", "key_id", "signature"],
        "properties": {
          "signature_algorithm": { "type": "string", "minLength": 1 },
          "key_id": { "type": "string", "minLength": 1 },
          "signature": { "type": "string", "minLength": 1 }
        }
      }
    ]
  },
  "extensions": {
    "type": "object",
    "additionalProperties": true
  }
}

```

The object fields have the following operational meanings:

Field	Reader obligation
tmpa_version	select the compatible Core object-schema line; unknown major versions are not silently downgraded
id	index canonical identity and detect same-ID conflicting content
type	resolve one versioned type-registry entry
governed_work.id	group objects that govern the same work item
governed_work.primary_carrier_id	identify the single stable carrier to which follow-on evidence must resolve
governed_work.parent_id/thread_id	preserve work derivation and a session-independent collaboration thread; the thread does not replace the parent relation
stream	establish attributable local order without using timestamps
creator and role	evaluate an authority claim against active assignments; these fields do not create authority
lifecycle	identify the profile and declared state; transition, when present, supplies explicit from/action/to evidence
references	construct typed ordering or non-ordering links according to the relation registry
claims	represent inspectable assertions and their evidence-object set; presence does not establish the claim
risk	represent a profile-defined risk level and human-approval requirement; it does not itself grant approval
content	carry the governed payload in the declared media type
integrity	identify the canonicalization and verification procedure for covered bytes
extensions	contain all profile-specific additions; unknown extensions are processed only under the declared profile

The primary-carrier object uses its own id as governed_work.primary_carrier_id. Every other object

for the same work item repeats that carrier identifier. A lifecycle-transition document type SHALL require `lifecycle.transition`; non-transition types MAY omit it. The type registry, rather than the generic single-object schema, enforces that conditional requirement.

Schema processors used for C01 SHALL implement JSON Schema Draft 2020-12 format assertion for `created_at`. A processor that treats `date-time` as annotation-only is insufficient. The linked S1.0 machine-readable artifact is the normative schema byte sequence; the embedded rendering above SHALL remain semantically identical to it.

S1.0 machine-readable artifact	SHA-256
Governance Object Schema	a2829cd7149c3054a52886365f2293a23106b636b0c52799739bfabdab1ff4fa
Lifecycle Profile Schema	481a61ac2485bbaf15d90e9c5a255ad9ce6a55971190f0fe404856be4b10f993
Reader Result Schema	4527df7096fe840b85b245e50d5cea576ff359d50a54d17c8873a7b4f458d431
Conformance Result Schema	4b1ecef83e62d2aa1aff0e79a0cd0ea0a85fbc14a426d5fe873ab40aefdc2fe

The `lifecycle-profile` schema requires explicit `acceptance`, `work_graph`, `risk_policy`, and `failure_model` sections in addition to `states`, `actions`, `transitions`, and `recovery rules`. S1.0 additionally requires the `risk policy` to identify permitted approval-object types and whether the approver must be independent. These sections make FCoP-derived collaboration-cycle semantics inspectable without binding TMPA to the FCoP reference implementation or to CodeFlowMu.

The `lifecycle.state` field records the state declared for this immutable object at publication. It is not a mutable current-state field. The current authoritative lifecycle state is reconstructed from the valid object set, accepted transition evidence, and the applicable lifecycle profile.

A canonicalization profile must define the exact representation covered by the digest and, when signatures are used, the exact representation covered by the signature. It must also define how self-referential integrity fields are excluded or normalized. TMPA Core S1.0 requires that this profile be declared; it does not prescribe one universal byte-level canonicalization algorithm.

Schema validity is necessary but insufficient for acceptance into an authoritative governance view. A reader must still evaluate identifier uniqueness, type rules, stream order, authority, lifecycle legality, references, digest verification, and any applicable signature policy.

4.2 Canonical Textual Encoding Profile

A canonical textual representation is not defined merely by choosing JSON, YAML, or Markdown. A versioned canonicalization profile is complete only when independent implementations can produce the same covered byte sequence from the same governed content.

At minimum, the profile defines:

- character encoding and Unicode normalization form;
- line-ending normalization;
- deterministic field and collection ordering where order is not semantically significant;
- whitespace, escaping, quoting, and delimiter rules;
- numeric representation, including exponent, sign, and precision rules;
- timestamp syntax, timezone requirements, and fractional-second normalization;
- the distinction among `absent`, `null`, `empty-string`, and `empty-collection` values;
- whether textual bodies, including Markdown whitespace and line endings, are covered verbatim or normalized;
- how attachments and external evidence are represented through media type, byte length, content digest, and optional locator metadata;
- whether extension fields are covered by the digest and signature, and how unknown extensions are ordered or rejected;
- the schema, type-registry, and canonicalization-profile versions bound to the object;
- how self-referential integrity fields are excluded or normalized before hashing or signing.

Semantic equivalence is insufficient for integrity verification. Two objects that a human considers equivalent may produce different digests when they differ in Unicode form, line endings, numeric spelling, timestamp precision, field order, or extension treatment. C08 and C11 are therefore meaningful only relative to the same declared canonicalization profile and version.

An external attachment or mutable URL is not authoritative evidence merely because it is referenced. A profile claiming integrity for external content records a content digest and the metadata required to identify the covered bytes. A locator may assist retrieval, but the locator alone does not preserve the referenced evidence.

4.3 Aggregation and Governance-Reconstruction Procedure

Given a finite unordered collection O of source candidates and a fixed rule profile P , a conforming implementation applies two stages.

AGGREGATE(O):

1. Discover each source artifact as a source candidate without assigning governance meaning to discovery order, filesystem order, transport order, or modification time.
2. Preserve source identity and bytes, then parse the candidate envelope. Retain parse failures as source evidence and deterministic diagnostics.
3. Canonically index object ids, writer streams, sequence numbers, task carriers, references, lifecycle locations, and integrity metadata.
4. De-duplicate byte-identical source observations without deleting their provenance; retain non-identical same-id variants as separate candidates.
5. Return the source-preserving canonical candidate set C and aggregation diagnostics.

RECONSTRUCT(C, P):

1. Validate object shape and document type. Retain invalid candidates for diagnostics, but exclude them from the authoritative candidate set.
2. Recompute each digest under the declared canonicalization profile. Retain digest-mismatched objects as evidence of an integrity failure, but exclude them from the intact authoritative candidate set.
3. Verify signatures when present. Record verification status and apply P 's acceptance policy. An unverifiable signature never establishes authenticated integrity.
4. Group intact candidates by object id.
 - a. Same id and same canonical content: project one object while retaining every contributing 'source_id' in Unicode code-point order.
 - b. Same id and different canonical content: quarantine all variants from the authoritative graph and emit a duplicate-id conflict.
5. Validate one-primary-carrier rules for task-oriented profiles and verify that follow-on reports, reviews, decisions, and corrections reference the carrier rather than creating ambiguous mutable task copies.
6. Group accepted objects by writer stream and order by sequence. Detect duplicate sequence numbers and sequence gaps without inventing missing objects or using timestamps as substitutes.
7. Build directed dependency edges only for relation types that P declares as ordering or lifecycle dependencies. Preserve other references as non-ordering graph links.
8. Validate role authority, separation of duties, lifecycle source state, transition legality, preconditions, and required evidence. Invalid actions remain observable but do not change authoritative state.
9. Detect missing references and cycles prohibited by P . Quarantine only the affected prohibited-cycle subgraph; retain unaffected valid objects.
10. Construct the accepted partial-order process and governance graph from within-stream sequence edges and profile-defined cross-stream dependencies. Preserve unrelated nodes as concurrent and incomparable.
11. When a canonical linear serialization is required for interchange or display, generate a deterministic topological serialization. Use object-id lexical order only as a tie-breaker among incomparable nodes; do not add those tie-breaker relations to the governance graph.
12. Project task, message, workflow, responsibility, lifecycle, review, approval, recovery, and audit views.
13. Canonically normalize both the reconstructed graph or view and the issue set, then return them together.

The composed operation is:

$$R_P(A(O)) = (G, I)$$

where $A(O)$ is the source-preserving canonical candidate set, G is the reconstructed partial-order process and governance graph, and I is the canonical issue set. The procedure is governed by the following invariants:

- source discovery, enumeration, and arrival order do not affect the final canonical output for the same source set;
- aggregation preserves source evidence and does not silently decide governance conflicts;
- one governed task has one stable primary carrier under a task-oriented profile;
- every published object has one writer and belongs to one local serial stream;
- timestamps do not override stream sequence or explicit dependency;
- no cross-stream order is invented when the profile defines no causal or lifecycle relation;
- any canonical linear serialization is a representation of the graph, not additional governance truth;
- invalid, disputed, or rejected evidence is not silently erased;
- a conflict is resolved only by a new authorized governance object;
- the same canonical source set and fixed profile produce the same aggregated candidate set, reconstructed view, and issue set;
- a partial or disputed view remains distinguishable from an authoritative view.

Delayed evidence changes the available source set and may change the current view legitimately. A task-only set may be partial; the same process may become authoritative or disputed after reports, reviews, or decisions arrive. Determinism requires equal output for equal source sets, not identical output across different stages of an asynchronous process.

4.4 Conflict and Validation Handling

A reader applies the following behavior consistently with the normative requirements in Chapter 9:

Condition	Required behavior
Schema or type invalid	retain source for diagnostics, exclude from authoritative reconstruction, emit validation issue
Digest mismatch	retain source as integrity-failure evidence, exclude from intact authoritative set
Signature absent	permit TMPA Core processing when other requirements pass; do not claim authenticated integrity
Signature unverifiable	emit signature issue and do not place object in an authenticated view
Same ID, same canonical content	safely de-duplicate for projection while retaining every contributing source identity
Same ID, different canonical content	quarantine all variants from authoritative graph and emit critical duplicate-ID issue
Missing reference	retain object in a partial view and emit unresolved-reference issue
Stream sequence gap	mark stream incomplete; do not infer the missing object or transition
Duplicate stream sequence	retain conflicting objects, mark affected stream non-conformant, and keep affected state partial or disputed
Timestamp conflict	ignore timestamp as authority; use sequence and profile-defined dependencies
Illegal lifecycle transition	retain attempted transition as evidence, do not alter authoritative state, and emit lifecycle issue
Unauthorized role action	retain attempted action as evidence, do not apply it, and emit authorization issue
Prohibited cycle	quarantine affected subgraph, report cycle, and continue reconstructing unaffected valid objects
Parallel contradictory reviews	preserve every valid review until an authorized resolution object exists

5. Threat Model and Trust Assumptions

5.1 Protected Properties

TMPA is designed to protect the following properties:

- attribution of objects to declared creators and roles;
- detection of object modification after publication;
- visibility of lifecycle and authority violations;
- preservation of conflicting evidence;
- reconstruction of governed work after runtime interruption;
- separation of execution, review, and approval responsibilities.

5.2 Trust Roots

A deployment must identify its trust roots. These may include:

- an identity provider;

- a role-assignment authority;
- a key registry or enterprise PKI;
- a trusted storage boundary;
- a human administrator or governance board;
- a trusted protocol validator.

TMPA does not create a trust root merely by writing a role name into a document. A role claim is valid only when the implementation can verify that the creator was authorized to act under that role at the relevant time.

5.3 Threats Considered

A conforming implementation should consider:

- identity impersonation;
- unauthorized role claims;
- object tampering;
- replay of previously valid objects;
- illegal lifecycle transitions;
- omission of required evidence;
- fabrication of false but well-formed evidence at initial publication;
- equivocation through conflicting objects;
- compromised tools or connectors;
- prompt injection that causes unauthorized actions;
- deletion or withholding of evidence;
- clock skew and timestamp manipulation;
- automatic remediation triggered by an incorrect, stale, or adversarial audit finding;
- transitive delegation that expands rather than attenuates authority;
- privilege aggregation in which individually permitted evidence or capabilities combine into an unauthorized result;
- stale, replayed, or insufficiently revoked delegation evidence;
- path-composition risk in which individually permitted actions form an unauthorized sequence;
- nominally independent reviewers controlled by the same model, controller, credential, host, or administrative principal.

5.4 Malicious Participants and Storage-Surface Compromise

TMPA Core does not assume that every participant is honest. It preserves attribution, conflicting objects, rejected transitions, and validation issues so that misconduct can be detected or investigated.

False-but-well-formed evidence at publication time

TMPA distinguishes **fabrication at publication time** from post-publication tampering. A malicious, compromised, or mistaken participant may publish a schema-valid, digest-consistent, lifecycle-legal, and even correctly signed REPORT, REVIEW, or DECISION whose factual claims are false. Core validation can establish structural validity, continuity, declared authority, and integrity of the published bytes; it cannot infer semantic truth from those properties. Authenticated Governance Conformance strengthens identity and authorization evidence, but a valid signature still proves origin and integrity rather than correctness.

Factual assurance therefore requires a declared evidence profile appropriate to the claim: tool receipts, externally verifiable outputs, reproducible execution, attached test results, independent data sources, cross-role verification under genuinely separate security principals, or human approval. When executor and reviewer share the same compromised controller, credential, evidence source, or administrative principal, nominal separation of duties may produce correlated fabrication rather than independent assurance. TMPA can preserve provenance, contradiction, and later correction; it does not detect covert collusion or guarantee that an initially published claim is true.

FCoP also exposes a protocol-specific attack surface because the filesystem is part of the protocol boundary. A participant with direct write permission may attempt to create an object inside `_lifecycle/done/`, alter a published artifact, remove evidence, replay a previously valid file, or create a path/event mismatch without using the authorized lifecycle operation. File presence alone must therefore not be treated as proof of validity.

A conforming reader should distinguish at least three cases:

1. **unauthorized insertion:** an artifact appears in a lifecycle location without a valid creator, role assignment, predecessor state, or transition record;
2. **post-publication mutation:** the content no longer matches its recorded digest or signature;
3. **state-evidence divergence:** the lifecycle path, transition history, references, and expected paired artifacts do not agree.

These attacks can be detected only to the extent that the deployment protects or independently verifies identity bindings, integrity records, append-only events, and storage history. If an attacker can both rewrite artifacts and replace every trusted integrity, identity, and audit record, the local filesystem view cannot establish a truthful history. Stronger deployments may add restricted write permissions, append-only or versioned storage, remote notarization, transparency logs, replication, or cryptographic signatures. These are deployment controls beyond the minimal FCoP filesystem profile.

TMPA Core does **not** provide Byzantine consensus. If the identity provider, role authority, key registry, trusted storage boundary, and validators are all compromised, TMPA cannot guarantee a truthful history. A deployment requiring Byzantine fault tolerance must add an external consensus, replication, notarization, or transparency-log mechanism.

An inspect-only audit profile narrows one failure impact: a compromised inspector can generate misleading findings or recommendations, but the audit function does not directly mutate governed business state. This is not complete protection. A human or external agent may still execute a harmful recommendation, so recommendation provenance, review, approval, and execution evidence remain required.

5.5 Levels of Role and Identity Separation

TMPA distinguishes logical responsibility separation from security-domain separation. A deployment may implement one or more of the following levels:

1. **prompt-level role separation:** participants receive different natural-language role instructions, but may share the same runtime, credential, and storage permissions;
2. **process-level identity separation:** distinct agent or process instances have stable runtime identities and separately attributable sessions;
3. **credential-level separation:** participants use distinct credentials, keys, or delegated authorization grants whose scopes can be validated and revoked independently;
4. **host-level isolation:** operating-system accounts, containers, sandboxes, or mandatory access controls prevent one participant from modifying another participant's protected evidence directly;
5. **administrative-domain separation:** independent organizations, trust roots, audit services, or transparency systems reduce the risk that one administrator can rewrite every relevant record.

Prompt-level separation can support workflow clarity, but it is not a security boundary. Process-level identity improves attribution but does not prevent credential sharing or cross-process storage mutation. Strong claims of independent review, authenticated responsibility, or tamper resistance require deployment controls at the credential, host, or administrative-domain levels appropriate to the threat model.

Under the minimal FCoP profile, a filename, filesystem owner, role document, or frontmatter sender is **declarative attribution**. It becomes verified attribution only when the deployment validates the binding among the acting process, credential or key, active role assignment, and protected write boundary. Several logical roles controlled by one model, one service account, or one unrestricted host must not be described as independent security principals merely because their prompts or filenames differ.

A verified delegation chain additionally distinguishes the delegating principal, delegated task or intent, granted capability set, attenuation rules, temporal validity, execution count or revocation conditions, and

each downstream re-delegation. A prior TASK, REPORT, or role label does not authorize a new action unless the active identity and authorization profile explicitly recognizes it as current delegation evidence.

5.6 AI Agent Identity

An AI agent should not be treated as a self-authenticating legal identity. FCoP nevertheless requires the agent to receive an explicit operational identity that it can read: its role, team context, responsibility boundary, and current work scope. The authority behind that operational identity still derives from a human or organizational principal, deployment identity, role-assignment authority, runtime credential, and policy scope.

A useful identity record distinguishes:

- organizational principal;
- human authorizer;
- agent instance;
- model or runtime version;
- active role;
- delegated permissions;
- credential or key identifier.

This distinction prevents an agent's actions from being attributed only to a borrowed human or service account.

5.7 Security Claims

An implementation must state which claim it supports:

Claim	Minimum requirement
textual traceability	persistent canonical objects and references
tamper detection	deterministic digest verification against preserved or trusted integrity metadata
authenticated integrity	verified signature and trusted key binding
authorization enforcement	validated role assignment and action policy
semantic claim verification	claim-specific evidence, reproducible outputs, or independent domain verification outside TMPA Core
non-repudiation	legal and cryptographic profile beyond TMPA Core
Byzantine resilience	external consensus or equivalent mechanism

An implementation must not claim a stronger property than its deployed controls provide.

6. Lifecycle and Authority Evaluation

6.1 Required Registries

An implementation profile publishes versioned lifecycle, role, and relation registries. A lifecycle-registry entry contains: profile identifier and version; state set; initial and terminal states; action set; legal from/action/to tuples; roles permitted for each action; required references and preconditions; separation-of-duty rules; and any authorized reopening or recovery rules. A role-registry entry contains: role identifier; assignment-object type; permitted document types and lifecycle actions; scope dimensions; incompatible roles; assigning authority; and revocation semantics. A relation-registry entry states whether the relation is ordering, non-ordering, required, or acyclic.

Registry bytes are inputs to reconstruction. Their versions and digests are therefore part of the reader input contract and conformance report; changing a registry while retaining its identifier does not produce the same fixed profile.

6.2 Transition Evaluation Order

For a candidate transition *x*, profile *P*, canonical candidate set *C*, and current reconstructed state *s*, evaluation follows this fixed order:

EVALUATE_TRANSITION(*x*, *s*, *C*, *P*):

1. validate object schema, type rule, identity, and integrity
2. resolve the governed work item, primary carrier, and lifecycle profile
3. reconstruct the unique current state from accepted predecessor evidence
4. verify that *x.from* equals that current state
5. verify that (*x.from*, *x.action*, *x.to*) is a legal transition tuple
6. resolve an active role assignment and validate action scope
7. evaluate separation-of-duty rules and authorized exceptions
8. resolve required references, preconditions, and evidence
9. assign valid, invalid, or undetermined with canonical issues
10. apply *x.to* only when the transition judgment is valid

A proven violation—such as an illegal tuple, revoked authority, out-of-scope action, or prohibited role combination—produces *invalid*. Missing evidence—such as an unavailable assignment, unresolved predecessor, absent required reference, or ambiguous current state—produces *undetermined*. Only valid transitions change the authoritative lifecycle projection.

6.3 State Reconstruction

For each governed work item, the reader starts from the lifecycle profile's initial state after accepting a valid primary carrier. It then evaluates transition objects in the partial order established by writer-stream sequence and declared ordering dependencies. Wall-clock time does not select the next transition.

If two valid transition candidates consume the same source state and their effects are incompatible without an ordering relation or authorized resolution, the current state is *undetermined* and the view is *disputed*. The reader retains both branches and does not choose the latest arrival. A terminal state remains terminal unless the lifecycle registry explicitly defines an authorized recovery or reopening transition.

6.4 Authority Time and Revocation

The reader validates authority against assignment and revocation evidence applicable to the action. *created_at* alone is not a trusted authorization clock. A profile that makes time-sensitive authority claims defines the trusted time or sequence evidence used to determine whether an assignment was active.

When evidence proves that authority was inactive, the action is *invalid*. When the relevant authority interval cannot be determined, the action is *undetermined*. A profile also declares whether revocation is prospective or may invalidate a defined class of earlier actions; the reader does not invent retroactive effect.

7. Three-Valued Governance Logic

7.1 Judgment Domain

Every governed conclusion receives exactly one semantic judgment from $J = \{\text{valid}, \text{invalid}, \text{undetermined}\}$. *valid* means all mandatory acceptance conditions are established. *invalid* means at least one mandatory rule is proven violated. *undetermined* means neither acceptance nor violation can be established because required evidence is missing, conflicting, ambiguous, or awaiting authorized resolution.

The values describe governance knowledge under a fixed source set and profile. They do not assert the factual truth of the governed payload.

7.2 Primitive Classification Rules

Condition	Judgment	View reason
all required checks established and no governing issue remains	valid	authoritative
schema/type failure, digest mismatch, explicit authority denial, illegal transition, or proven separation-of-duty violation	invalid	rejected or quarantined
required reference, assignment, predecessor, stream element, or decision is missing	undetermined	partial or pending_human
multiple valid but incompatible claims lack an authorized resolution	undetermined	disputed
optional signature absent under Core	unchanged	unauthenticated assurance label
required authentication cannot be established under an authenticated profile	undetermined or invalid, as the published profile declares	unauthenticated or quarantined

7.3 Composition Rules

Mandatory conjunction ALL (a , b) and alternative satisfaction ANY (a , b) use the following truth table:

a	b	ALL (a , b)	ANY (a , b)
valid	valid	valid	valid
valid	undetermined	undetermined	valid
valid	invalid	invalid	valid
undetermined	valid	undetermined	valid
undetermined	undetermined	undetermined	undetermined
undetermined	invalid	invalid	undetermined
invalid	valid	invalid	valid
invalid	undetermined	invalid	undetermined
invalid	invalid	invalid	invalid

A required dependency with invalid judgment makes the dependent acceptance condition invalid; a required dependency with undetermined judgment propagates undetermined. Two incompatible valid claims do not cancel each other or become invalid; their unresolved combined conclusion is undetermined and disputed. A resolution changes the conclusion only when the resolution object is itself valid, authorized, and explicitly references the conflict it resolves.

Profiles may define domain-specific aggregations, but they publish their truth tables and may not map missing or conflicting mandatory evidence directly to valid.

7.4 Judgment and View Mapping

Judgment is semantic; view state explains the operational reason. valid maps to authoritative. invalid maps to rejected for an action or quarantined when evidence or a subgraph is excluded. undetermined maps to disputed, partial, or pending_human according to the canonical issue causes.

When one subject has several causes, all causes remain in the issue set. If one primary view label is required, the ordering is quarantined → rejected → disputed → partial → pending_human → authoritative. Authentication remains a separate assurance status and does not create a fourth semantic judgment.

8. Reader Input and Output Contract

8.1 Input Bundle

A deterministic reader invocation fixes:

- Core object-schema version and digest;
- conformance-profile identifier, version, and digest;

- type, lifecycle, role, relation, integrity, and canonicalization registries with versions and digests;
- the finite source-candidate multiset, where each candidate has a stable `source_id`, media type, exact bytes, and byte digest;
- declared trust roots and authentication policy;
- reader implementation identifier and version;
- canonical output format version.
- every implementation extension and whether it affects canonical semantics.

Two invocations are comparable for C11 only when these inputs are equal. Environment-specific locators, discovery timestamps, log order, memory addresses, and localized diagnostics are not canonical inputs.

8.2 Canonical Result

The reader emits one result envelope with at least:

```
{
  "core_version": "S1.0",
  "output_version": "1",
  "profile": {},
  "reader": { "id": "<id>", "version": "<version>" },
  "source_set_digest": "sha256:<hex>",
  "judgment": "valid | invalid | undetermined",
  "view_state": "authoritative | rejected | quarantined | partial | disputed | pending_human",
  "nodes": [],
  "edges": [],
  "issues": []
}
```

Each node and edge SHALL contain a stable identifier and its source-object identifier. Each issue SHALL contain a stable `issue_id` and `source_id`; it records `source_object_id` when parsing produced one. Nodes SHOULD additionally record canonical digest, governed-work ID, primary-carrier ID, type, stream position, judgment, view state, and retained source IDs. Edges SHOULD record relation and ordering semantics. Issues SHALL record code and severity and SHOULD record the affected judgment, normative rule, and deterministic parameters.

Core issue codes are: `SCHEMA_INVALID`, `UNKNOWN_TYPE`, `INTEGRITY_MISMATCH`, `SIGNATURE_UNVERIFIED`, `DUPLICATE_ID_CONFLICT`, `PRIMARY_CARRIER_CONFLICT`, `STREAM_DUPLICATE_SEQUENCE`, `STREAM_GAP`, `AUTHORITY_UNDETERMINED`, `AUTHORITY_DENIED`, `SOD_VIOLATION`, `LIFECYCLE_UNDETERMINED`, `ILLEGAL_TRANSITION`, `MISSING_REFERENCE`, `PROHIBITED_CYCLE`, `UNRESOLVED_CONFLICT`, `CLAIM_EVIDENCE_MISSING`, `ACCEPTANCE_UNDETERMINED`, `HUMAN_APPROVAL_REQUIRED`, `CHILD_WORK_OPEN`, `RECIPROCITY_MISSING`, and `STATE_EVIDENCE_CONFLICT`. Profiles namespace additional codes and do not redefine Core codes.

8.3 Canonicalization and Ordering

The source-set digest is computed from the deterministically sorted list of (`source_id`, `byte_digest`) pairs under the declared output profile. Nodes sort by (`id`, `source_object_id`). Edges sort by (`source_id`, `relation`, `target_id`, `id`). Issues sort by (`severity`, `code`, `object_id`, `relation`, `target_id`, `issue_id`), using severity order critical, error, warning, info; absent tuple fields are empty strings.

An `issue_id` is derived from the canonical tuple (`code`, `object_id`, `relation`, `target_id`, `profile_digest`) under the output profile. Human-readable messages, stack traces, local paths, and execution timestamps are excluded from canonical equality. Object keys, subjects, retained source IDs, nodes, edges, and issues use Unicode code-point order after profile-defined normalization; locale-sensitive collation is not canonical.

The canonical result serialization is byte-stable for equal fixed inputs. Non-canonical logs and user-interface ordering may vary, but they do not alter the result envelope used for C11.

8.4 Failure and Partial Output

The reader returns a canonical result and issue set even when some candidates are malformed or a sub-graph is invalid. It may fail the entire invocation only when the fixed profile, schema, registry bundle, or output canonicalization contract cannot be loaded or verified. Such an invocation failure is distinct from an invalid governance judgment and is reported as a conformance-run error.

9. Normative TMPA Core Specification

9.1 Normative Language

The terms **MUST**, **MUST NOT**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** define conformance requirements. Mandatory requirements are expressed with **MUST**, **MUST NOT**, **SHALL**, or **SHALL NOT**.

Descriptive examples, implementation observations, and future-work statements outside this chapter do not create additional TMPA Core requirements unless they are incorporated explicitly by a named conformance profile.

9.2 Object Requirements

Every governance object **MUST**:

- conform to the TMPA Core schema and its published document-type definition;
- have a globally unique identifier within its governance domain;
- have exactly one declared creator identity;
- identify exactly one responsible role;
- identify one stream and one positive sequence number;
- identify one document type;
- identify one governed work item and exactly one primary carrier identifier;
- identify one lifecycle profile and declared state;
- contain canonical textual content;
- contain a references array, which **MAY** be empty;
- contain integrity evidence.

A conforming validator **SHALL** enforce the declared date - time format for `created_at`; treating the format only as an annotation is insufficient for C01 conformance.

An object type that records a lifecycle transition **SHALL** include one complete `from`, `action`, and `to` tuple. A non-transition type **SHALL NOT** use that tuple to create an implicit state change.

A published object **SHALL** be immutable. A correction, rejection, supersession, rollback, or resolution **SHALL** create a new object or transition record and **SHALL** preserve the earlier evidence.

Schema validity alone **SHALL NOT** be interpreted as proof of identifier uniqueness, role authority, lifecycle legality, reference validity, digest correctness, or authenticated identity.

A task-oriented profile **SHALL** define one stable primary carrier identifier for each governed work item. Subsequent acceptance, report, review, decision, correction, and recovery objects **SHALL** reference that carrier or a profile-defined successor relation rather than create ambiguous mutable copies of the same task.

Every published governance object **SHALL** have one writer. A different participant **SHALL** respond through a new attributable object or transition record and **SHALL NOT** modify the published content of another writer's object.

9.3 Type Registry Requirements

A conforming implementation **SHALL** publish its document-type registry.

The registry SHALL have a stable identifier, version, and byte digest. A reader SHALL bind its result to that exact registry revision.

Each type definition SHALL specify:

- permitted creator roles;
- required fields;
- permitted reference relations;
- the applicable lifecycle profile;
- whether the type requires a lifecycle transition tuple;
- validation rules.

A document SHALL NOT serve simultaneously as its own independent review or approval unless the implementation profile permits a recorded exception and the exception identifies its approving authority.

9.4 Role Requirements

A role claim SHALL be validated against an authoritative role assignment active for the relevant object and action.

A participant SHALL NOT perform a protected action outside the active scope of its role.

A deployment claiming separation of duties SHALL define and enforce incompatible role combinations for the same governed result.

Role assignment, revocation, delegation, and separation-of-duty exceptions SHOULD themselves be represented as governance objects.

Role and authority evaluation SHALL follow the order in Section 6.2. Proven denial or an out-of-scope action SHALL be invalid; missing or ambiguous assignment evidence SHALL be undetermined.

9.5 Stream Requirements

Every stream SHALL have a stable stream identifier.

Every published object SHALL belong to exactly one writer stream. A conforming profile SHALL preserve the local publication order of that writer and SHALL NOT require multiple writers to co-author one mutable object.

Independent streams MAY progress asynchronously and SHALL NOT be required to advance in lockstep. The absence of a new object in one stream SHALL NOT prevent unrelated streams from progressing when the profile defines no dependency.

Sequence numbers SHALL be positive integers and SHALL be unique within a stream.

A reader SHALL report a duplicate sequence number as a stream-integrity error and SHALL preserve every conflicting candidate for inspection.

A reader SHALL report a sequence gap and SHALL NOT invent the missing object, infer its content, or use timestamps to replace the missing sequence position.

Wall-clock timestamps SHALL NOT be the sole authoritative ordering mechanism.

A reader SHALL NOT infer an authoritative order between objects in different streams unless the applicable profile defines a cross-stream causal, lifecycle, or dependency relation. Objects without such a relation SHALL remain concurrent or incomparable in the reconstructed governance graph.

9.6 Lifecycle Requirements

Every lifecycle profile SHALL define:

- states;

- an initial state;
- terminal states;
- actions;
- legal transitions;
- authorized roles;
- preconditions;
- required evidence.

Every lifecycle profile SHALL have a stable identifier, version, and byte digest. A reader SHALL validate transitions in the order defined by Section 6.2, reconstruct state as defined by Section 6.3, and apply only transitions judged valid. If the unique current state cannot be reconstructed, the candidate transition SHALL be undetermined and SHALL NOT alter authoritative state.

An illegal or unauthorized transition SHALL NOT alter the authoritative lifecycle state.

The attempted transition SHALL remain observable through a rejection, issue, alert, or equivalent profile-defined record unless the attempt cannot be captured by the deployment's stated threat model.

A terminal-state or archival operation SHALL preserve the objects and transition evidence required to reconstruct how that state was reached.

9.7 Reference Requirements

Every reference SHALL identify a relation type and target object identifier.

A profile SHALL define which reference types create ordering dependencies, which are non-ordering links, and which relation classes must be acyclic.

The relation registry SHALL have a stable identifier, version, and byte digest, and the reader result SHALL identify the exact revision used.

A missing target SHALL be reported. The referencing object MAY remain in a partial view, but the missing dependency SHALL NOT be treated as satisfied.

A reader SHALL quarantine the affected subgraph of a prohibited cycle rather than silently deleting an edge or selecting an arbitrary order. Unaffected valid objects SHOULD remain reconstructable.

9.8 Integrity Requirements

The canonicalization and digest algorithms SHALL be declared by a versioned integrity profile.

The integrity profile SHALL define the exact fields or bytes covered by the digest and, when applicable, the signature. It SHALL define how digest, signature_algorithm, key_id, and signature are excluded or normalized to avoid self-reference.

The profile SHALL also define character encoding, Unicode normalization, line endings, field ordering, whitespace and escaping, number and timestamp representation, absent-versus-null handling, textual-body treatment, attachment hashing, extension-field treatment, and the schema and profile versions bound to the canonical form. Unknown extension fields SHALL either be included deterministically in the covered representation or rejected; they SHALL NOT influence authoritative semantics while being silently excluded from integrity protection.

A reader SHALL recompute and verify the digest before accepting an object as intact. A digest mismatch SHALL be reported, and the object SHALL NOT enter the intact authoritative object set, although the source SHALL remain available as evidence of the failure.

When signature metadata is present, signature_algorithm, key_id, and signature SHALL be supplied as one complete group. The reader SHALL validate the signature, key status, and identity binding before claiming authenticated integrity.

An absent signature is permitted under TMPA Core. An unverifiable signature SHALL NOT be treated as valid evidence of authenticated integrity.

A deployment SHALL NOT present schema validity, digest validity, signature validity, role authorization, or lifecycle legality as proof that an object's factual claims are true. Semantic assurance requires a declared evidence and verification profile outside TMPA Core.

A deployment SHALL NOT claim resistance to an attacker who can alter both content and unanchored integrity metadata solely because a co-located digest is present. Such a claim requires authenticated integrity, externally anchored digests, trusted storage, or an equivalent declared control.

9.9 Aggregation and Reader-Reconstruction Requirements

A conforming source aggregator SHALL preserve source identity and content, discover candidates without using enumeration order as governance order, and produce a deterministic canonical candidate set for the reader. It SHALL NOT silently resolve conflicts, invent missing objects, or convert a transport or filesystem arrival order into an authoritative process sequence.

For the same canonical candidate set and fixed rule profile, a conforming governance reader SHALL:

- produce the same canonical reconstructed partial-order graph or view and issue set for every input permutation;
- preserve within-stream order, profile-defined cross-stream relations, and concurrency among incomparable objects;
- preserve source objects unchanged;
- preserve valid conflicting objects until an authorized resolution exists;
- exclude schema-invalid and digest-invalid objects from the authoritative object set while retaining their diagnostic evidence;
- report duplicate identifiers, sequence gaps, duplicate sequences, illegal transitions, unauthorized actions, missing references, prohibited cycles, and integrity failures;
- emit one semantic judgment—valid, invalid, or undetermined—for each governed conclusion and preserve the reason for that judgment;
- distinguish authoritative, partial, disputed, quarantined, and unauthenticated states where those distinctions apply;
- apply a deterministic order to conformance issues and serialized view elements.

The fixed rule profile SHALL include every input listed in Section 8.1. The reader SHALL emit the envelope defined by Section 8.2, use the Core issue codes and identifiers defined there, and apply the three-valued composition rules in Section 7.3. Canonical output ordering SHALL follow Section 8.3.

A deterministic topological serialization or display tie-breaker SHALL NOT be interpreted as a governance decision, truth priority, or additional cross-stream order.

A reader SHALL NOT use input arrival order, filesystem enumeration order, or wall-clock timestamp order to resolve a governance conflict.

An undetermined dependency SHALL propagate as undetermined to a dependent conclusion until an authorized resolution object satisfies the applicable profile. View classifications explain why the judgment was reached; they SHALL NOT replace or expand the three semantic values.

9.10 Recovery Requirements

A replacement participant SHALL be able to determine, from persistent governance objects and the applicable profile:

- the current authoritative or explicitly partial lifecycle state;
- the responsible role;
- unresolved requirements;
- referenced results, reviews, approvals, and rejections;
- integrity, authority, ordering, reference, and validation issues.

Recovery SHALL NOT require access to the previous participant's hidden chain of thought.

Execution-specific context not represented in governance objects MAY be unavailable; such absence SHALL be reported rather than guessed.

9.11 Governance Closure, Claims, and Human-Control Requirements

A work-oriented profile SHALL define the relation semantics for reports, issues, rejection, cancellation, follow-up work, review, acceptance, parent-child derivation, human approval, and archive authorization.

Lifecycle state and business acceptance SHALL be reconstructed separately. A terminal state, done label, physical archive, or executor completion claim SHALL NOT establish business completion by itself. When valid acceptance evidence is missing, the reader SHALL emit `ACCEPTANCE_UNDETERMINED`.

A governance-decision object SHALL remain orthogonal to a lifecycle review stage. An execution report SHALL NOT act as its own independent review, and a governance review SHALL NOT replace a required reciprocal response.

Every completion, failure, recovery, or acceptance claim SHALL identify a stable claim identifier, predicate, subject, and evidence-object identifier set. Missing required evidence SHALL emit `CLAIM_EVIDENCE_MISSING`, and the affected conclusion SHALL be undetermined.

Every accepted work object SHALL eventually have a report, issue, rejection, cancellation, or follow-up-work response. When the profile requires closure and the reader finds none, it SHALL emit `RECIPROCITY_MISSING` and SHALL NOT interpret silence as success.

Child work SHALL identify its parent explicitly. When a parent has an unfinished child, an unhandled blocked child, or a child without a reciprocal result, parent completion or acceptance SHALL emit `CHILD_WORK_OPEN` and remain undetermined.

A risk object requiring human approval SHALL remain undetermined / `pending_human` until the reader verifies a separate human-approval object whose type and relation are permitted by the risk policy, whose creator has an active assignment to a permitted approval role, and whose creator differs from the risk-object creator when the profile requires an independent actor. Agent self-approval, role labels without assignment evidence, missing approval, wrong object type, or unauthorized approval SHALL NOT satisfy the requirement and SHALL emit `HUMAN_APPROVAL_REQUIRED` or the applicable authority issue.

A profile SHALL publish failure-type and recovery-action registries. Failure and recovery objects SHALL reference the affected work; a recovery object SHALL also reference the triggering failure. A failure SHALL NOT be hidden or overwritten by a success response.

When current-state observations conflict with transition-history evidence, the reader SHALL emit `STATE_EVIDENCE_CONFLICT`, preserve both sources, and prevent the conflicting state from becoming the sole authoritative conclusion.

Inspection and governance-alert objects MAY emit findings and suggested plans, but they SHALL NOT be interpreted as an applied repair, lifecycle transition, or business decision without separate execution evidence.

10. Conformance and Testability

Sections 9 and 10 preserve the clause identifiers used by the combined TMPA Draft V1.0 source. This stability allows conformance reports and fixtures to cite the same normative basis across the architecture paper, Core specification, and implementation reports.

10.1 Conformance Levels

TMPA defines three conformance levels:

1. **TMPA Core Conformance:** implements durable textual messages and state objects, primary-carrier rules, single-writer streams, asynchronous multi-stream progression, deterministic aggregation and governance reconstruction, type rules, roles, lifecycle, integrity verification, and recovery requirements.
2. **FCoP Profile Conformance:** satisfies TMPA Core through a documented projection to the published FCoP protocol, including its naming, lifecycle, atomic-transition, routing, and evidence rules. Passing tests for one FCoP reference-implementation package is implementation evidence only; it is neither installation of the protocol nor sufficient by itself for this conformance level.
3. **Authenticated Governance Conformance:** satisfies TMPA Core and validates creator identity through a trusted signature, key, and authorization profile.

None of these levels certifies the semantic truth of a participant's claim. Authenticated Governance Conformance can establish which verified principal published an authorized object; claim correctness still depends on the applicable evidence, review, tool-attestation, or domain-verification profile.

Conformance is a claim about a specified implementation version, profile version, fixture corpus, and result set. Product identity, repository ownership, package publication, or demonstration availability does not itself establish conformance.

10.2 Required Conformance Tests

A TMPA Core conformance suite SHALL include C01–C14. Each result SHALL identify its normative basis and preserve the actual output needed to reproduce the verdict.

ID	Test	Normative basis	Pass criterion
C01	Schema validation	4.1, 9.2, 9.3, 9.11	an object with missing required fields, wrong Core type/version, prohibited fields, malformed transitions, incomplete signatures, malformed claims, invalid risk enums, or invalid asserted date - time is excluded and produces <code>SCHEMA_INVALID</code> deterministically
C02	Primary carrier, work derivation, and single-writer immutability	9.2, 9.11	one stable task carrier remains identifiable; parent-child work round-trips and is not replaced by a thread; another writer cannot replace or co-edit a published object; correction/supersession uses new attributable evidence
C03	Duplicate object identity and source provenance	9.2, 9.9	same-ID candidates with different canonical content are retained and quarantined with a deterministic critical conflict; byte-identical observations project one node while retaining every contributing source ID
C04	Serial-stream continuity and asynchronous progress	9.5, 9.9	each writer preserves its local sequence; duplicate numbers and gaps are reported; unrelated streams can advance independently; no missing object is invented and arrival order does not change the final result for the same set
C05	Role authority	6.2, 9.4, 9.9	an action outside validated role scope produces <code>AUTHORITY_DENIED</code> and invalid; missing or ambiguous assignment evidence produces <code>AUTHORITY_UNDETERMINED</code> and undetermined; neither action is applied
C06	Lifecycle legality, state evidence, and business-acceptance separation	6.2–6.4, 9.6, 9.9, 9.11	an undefined transition produces <code>ILLEGAL_TRANSITION</code> ; ambiguous state or missing prerequisites produce <code>LIFECYCLE_UNDETERMINED</code> ; an observation contradicting reconstructed state produces <code>STATE_EVIDENCE_CONFLICT</code> ; completion without independent acceptance produces <code>ACCEPTANCE_UNDETERMINED</code> ; none fabricates business completion
C07	Separation of duties and human control	9.3, 9.4, 9.11	the same identity cannot execute and independently review the same result; self-approval, unassigned role labels, and wrong approval-object types remain pending; only a separate permitted object from an assigned, profile-authorized and, when required, independent approver satisfies the gate
C08	Integrity tampering	9.8, 9.9	changing covered content while preserving the original integrity metadata causes digest verification to fail; the object is retained as failure evidence but excluded from the intact authoritative set
C09	Missing reference and claim evidence	9.7, 9.9, 9.11	an unresolved target produces <code>MISSING_REFERENCE</code> ; missing evidence for a completion claim produces <code>CLAIM_EVIDENCE_MISSING</code> ; neither dependency nor claim is treated as satisfied
C10	Prohibited cycle	9.7, 9.9	the affected prohibited-cycle subgraph is quarantined and reported while unaffected valid objects remain reconstructable
C11	Aggregation and reconstruction determinism	8, 9.9	every tested enumeration, delayed-delivery permutation, and aggregation order of the same source set and complete fixed input bundle produces a byte-equivalent canonical result envelope, graph, and issue set; Unicode code-point ordering is locale-independent and unrelated cross-stream objects remain incomparable

| C12 | Conflict preservation | 9.9 | contradictory valid reviews remain visible and disputed until a new authorized resolution object is supplied | | C13 | Recovery and parent-child closure | 9.10, 9.11 | a fresh reader

reconstructs responsibility, lifecycle, unresolved dependencies, failure/recovery, and parent-child relations; open children produce CHILD_WORK_OPEN, without hidden runtime context | | C14 | Post-acceptance terminal-history preservation | 9.2, 9.6, 9.11 | terminal/archive state follows required acceptance and archive authority, and all task, report, review, acceptance, and transition objects needed for reconstruction remain available |

The tests are behavioral. An implementation MAY use different storage, indexing, or execution mechanisms, but the observable conformance result must satisfy the same criteria.

10.3 Executable Test-Case Contract

Each executable test case SHALL publish a machine-readable manifest containing:

- a stable test_case_id and exactly one C01–C14 criterion;
- the Core, object-schema, output-schema, Profile, and registry versions and byte digests;
- explicit prerequisites;
- a source-fixture list containing source_id, repository-relative path, media type, and byte digest;
- assertions containing a stable assertion ID, target, operator, expected value, and mandatory flag;
- the expected canonical result digest;
- the runner identifier, command, execution environment, and any permutation method or seed;
- repository-relative paths for stdout, stderr, canonical output, and supporting evidence.

The runner SHALL preserve the exact input manifest, canonical result, exit status, stdout, stderr, and execution-environment identity. A test SHALL NOT depend on an unpinned network response, wall-clock ordering, filesystem enumeration order, or undeclared mutable state.

```
{
  "test_case_id": "C06-illegal-transition-001",
  "criterion": "C06",
  "core_version": "S1.0",
  "inputs": [{"source_id": "transition-1", "path": "fixtures/C06/transition-1.json", "media_type":
    ↪ "application/json", "byte_digest": "sha256:<hex>"}],
  "assertions": [{"id": "state-unchanged", "target": "/nodes/work-1/state", "operator": "equals", "expected":
    ↪ "active", "mandatory": true}],
  "expected_result_digest": "sha256:<hex>",
  "runner": {"id": "tmpa-conformance", "version": "<version>", "command": "<command>"}
}
```

10.4 Verdict Algorithm and Conformance Claim

For each criterion, the runner SHALL assign exactly one verdict:

- **PASS:** every mandatory assertion executed and passed;
- **FAIL:** at least one mandatory assertion executed and failed;
- **PARTIAL:** at least one mandatory assertion executed and passed, none failed, and at least one did not execute;
- **NOT RUN:** no mandatory assertion executed, or a prerequisite prevented evaluation.

Infrastructure failure SHALL be recorded separately as run_state: error and produces NOT RUN, not PASS. The aggregate precedence is FAIL, PARTIAL, NOT RUN, then PASS: any FAIL makes the aggregate FAIL; with no FAIL, any PARTIAL makes it PARTIAL; with neither, any NOT RUN makes it NOT RUN; only all PASS yields PASS.

A product MAY claim **TMPA Core S1.0 Conformance** only when C01–C14 all report PASS against the same fixed input bundle and the complete evidence package is published. “No observed failure,” PARTIAL, NOT RUN, an earlier-Core result, or an unpublished result SHALL NOT be represented as full S1.0 conformance.

specified, implemented, demonstrated, and independently adopted describe evidence maturity and SHALL be reported separately from test verdicts. A demonstration by the authors does not establish independent adoption.

```
{
  "core_version": "S1.0",
  "implementation": {"id": "<id>", "version": "<version>"},
}
```

```

"criteria": [{"id": "C01", "verdict": "PASS", "manifest_digest": "sha256:<hex>", "result_digest": "sha256:<hex>"}],
"aggregate_verdict": "PASS | FAIL | PARTIAL | NOT RUN",
"evidence_level": "specified | implemented | demonstrated | independently_adopted"
}

```

10.5 Test Fixtures and Result Reporting

The conformance package SHOULD publish:

- valid TMPA Core object fixtures;
- valid FCoP TASK, REPORT, ISSUE, and REVIEW fixtures derived from the published schemas;
- invalid schema and format fixtures;
- illegal and unauthorized transition fixtures;
- unauthorized role and separation-of-duty fixtures;
- broken digest and signature fixtures;
- duplicate-ID, duplicate-sequence, and sequence-gap fixtures;
- missing-reference and prohibited-cycle fixtures;
- parallel conflicting-review fixtures;
- controlled interruption and recovery snapshots;
- terminal-history or archival-preservation fixtures;
- expected canonical aggregated candidate sets, reconstructed process/governance graphs or views, and issue sets.

Each executable fixture set SHOULD identify:

- the TMPA schema version;
- the profile and rule-set version;
- the reader implementation and version;
- the canonicalization profile;
- input object identifiers and digests;
- expected accepted, partial, disputed, quarantined, and rejected identifiers;
- expected canonical view and issue-set outputs;
- permutation method, seed, and tested permutation count where applicable;
- runner command, execution date, and result.

A pipeline passes C11 only when the aggregator produces the expected canonical candidate set and canonical serialization of both the reconstructed process/governance graph or view and issue set matches the expected fixture for every tested enumeration and delivery permutation of the same final source set. Differences in non-canonical logging or internal data structure order do not constitute a failure unless they alter the canonical output.

10.6 Compliance Crosswalks

TMPA provides technical controls, not automatic legal certification. A deployment MAY map TMPA fields and tests to external requirements, including:

- organizational accountability;
- logging and record retention;
- human oversight;
- identity and authorization;
- separation of duties;
- incident investigation;
- evidence integrity.

The crosswalk SHALL identify whether each external requirement is fully supported, partially supported, unsupported, or outside TMPA scope. It SHALL also identify the external identity, policy, retention, and security systems on which the mapping depends.

A global interoperability profile and a jurisdiction-specific compliance profile are separate deliverables. For example, mapping FCoP artifacts to A2A tasks is an interoperability problem; mapping TMPA evidence to a

national or sectoral regulation is a compliance problem. The two may share governance objects, but neither should be claimed on the basis of the other.

11. Profile, Versioning, and Publication Rules

11.1 Core and Profile Separation

TMPA Core defines portable governance semantics. A profile MAY add document types, lifecycle profiles, storage mappings, identity bindings, integrity policies, or application-specific rules. A profile MUST identify its version, MUST state which Core conformance level it claims, and MUST NOT weaken a Core MUST while continuing to claim the affected conformance level.

A profile-specific artifact is not automatically a canonical Core object. The profile MUST define a deterministic projection from source artifacts into source candidates, canonical objects, governance graph nodes and edges, and issue-set entries.

11.2 Versioning

Changes that alter required fields, authority semantics, lifecycle legality, canonicalization, reader output, issue classification, or C01–C14 pass criteria require a new Core version. Editorial clarification that does not alter observable behavior MAY retain the current Core version but SHOULD be recorded in a changelog.

Conformance reports MUST identify the exact Core version, profile version, reader implementation, canonicalization profile, fixtures, source revision, and execution environment.

11.3 Publication and Evidence Boundary

Publication of a specification establishes the **specified** evidence level. Executable code may establish **implemented** behavior for tested paths. A bounded run may establish **demonstrated** behavior. None of those establishes independent adoption or validation without an external implementation, rerun, or organizational reliance.

The first author-produced C01–C14 corpus is maintained as a separate empirical artifact rather than embedded in this Core specification. Product verdicts and case evidence belong in the implementation and case report; the normative criterion meanings remain defined here.

11.4 S1.0 Release and Evidence Record

The 2026-08-10 evidence-candidate audit froze the English and Chinese documents, all four machine-readable schemas, lifecycle Profile, canonicalization Profile, Reference Reader, and C01–C14 fixtures as one versioned input bundle at commit 942cbb097eb3d662662f96a2269818ec9d7ca2ed. S1.0 promotes the reviewed S0.6 normative design without adding a new governance concept.

The author-produced S1.0 Reference Reader passes all fourteen S1.0 fixtures. The frozen candidate's product baseline remains NOT RUN and SHALL NOT be rewritten. A later exact-version external registration records CodeFlowMu V1.8.0 commit c1e1f724293e8048fc3a956b6f6df8cf83f54830 executing its product GovernanceReader.readSync against that bundle with **14 PASS / 0 PARTIAL / 0 NOT RUN / 0 FAIL**. Its input-bundle digest is sha256:f98764987760cdc8ac356b1265fc98485f33345e7d6ffc8575ccb059ddd34daa; its result digest is sha256:0f0f642449db1853371861751a7a8ea36dce00013f53e32012a5e4dae45f4c39.

The V1.8.0 package SHA-256 is 9b92b06c3e46c8019362f55075be4ed066cb7a9c0d9859945b6c3b7de8840d04. It establishes only author-run demonstrated behavior for the exact product revision and input bundle. It does not establish independent validation, third-party certification, semantic truth,

universal conformance, hallucination elimination, or independent adoption. I0.8 and the CodeFlowMu V1.6.0 package remain predecessor S0.6 evidence and SHALL NOT be relabeled.

11.5 S0.5 FCoP-Derived Historical Baseline

S0.5 derived lifecycle-state/business-acceptance separation, parent-child work, completion claims, role capability layers, risk and human-approval gates, reciprocity, failure/recovery actions, inspection findings, and drift handling from the complete, version-pinned FCoP protocol source set. FCoP remains a protocol and reference Profile; the `fcop` and `fcop-mcp` Python packages remain reference implementations rather than the protocol itself. S0.4/I0.5, S0.5/I0.6, and S0.5/I0.7 retain their exact historical meanings in Git history and their published evidence packages.

Appendix A. Historical Source Traceability (Informative)

Core specification content	Historical source section	Current treatment
terminology and representation stages	Section 1.5	retained and renumbered
governance objects, roles, lifecycle, streams, aggregation, integrity	Sections 4.1–4.7	retained and renumbered
canonical schema, encoding, reader algorithm, conflict handling	Sections 6.1–6.2 and 6.5–6.6	retained; FCoP mapping and product example excluded
threat model and security boundaries	Chapter 8	retained and renumbered
normative Core clauses	Chapter 9	retained with original 9.x identifiers
conformance levels and C01–C14	Sections 10.1–10.2	retained with original identifiers
fixture and result-reporting requirements	Section 10.5	retained; current product baseline removed
compliance crosswalk boundary	Section 10.6	retained

The Architecture Paper may summarize this specification but cannot redefine it. The Implementation Case Report may provide evidence against these clauses but cannot change their meaning. The historical combined draft records provenance only and has no current editorial or normative authority. All S1.0 and later normative changes are maintained directly in this GitHub Core Specification and represented by Git history.

Appendix B. FCoP Source Crosswalk (Informative)

S1.0 concern	Version-pinned FCoP source	TMPA Core treatment
protocol object, document and event vocabulary	<code>spec/fcop-v3-spec.md</code> and <code>spec/fcop-v3-spec.zh.md</code> ; repository tag <code>v3.2.5</code>	projected into governance objects, typed references, writer streams, and source-preserving Reader input declared capability and enforced authority remain separate; acceptance and separation-of-duty decisions require attributable evidence
role boundaries and collaboration-cycle rules	<code>AGENTS.md</code> , rules line 3.2.5	informs S1.0 object/profile schemas but does not replace TMPA Core schema validation
machine-readable carriers and validation	<code>spec/schemas/</code>	formalized as lifecycle state, business acceptance, failure/recovery actions, inspection findings, and deterministic history reconstruction
lifecycle, atomic transition, recovery and audit decisions	FCoP specification and applicable ADRs	represented by <code>governed_work.parent_id</code> , parent-child rollup, and <code>CHILD_WORK_OPEN</code>
parent-child work derivation and closure	FCoP v3.2.5 parent protocol surface	

S1.0 concern	Version-pinned FCoP source	TMPA Core treatment
executable software	fcop and fcop-mcp packages	treated only as the FCoP reference implementation; package tests are implementation evidence, not the protocol itself
downstream use	CodeFlowMu and bounded WP-13 evidence	treated as application evidence in the Implementation Case Report, never as proof of the theory or definition of the protocol; XiaoDian AI is retained as author-reported lineage only and is excluded from evaluated evidence

The crosswalk is a traceability aid, not an incorporation by reference. Where FCoP and TMPA use different abstractions, this Core specification controls TMPA meaning; where an application or reference implementation diverges from its protocol source, the divergence is reported as implementation evidence rather than silently changing either specification.