PRAXIS AGENT ID

The Agentic Identity Document (AID) Standard

Anchored, versioned, standards-graded identity for autonomous AI agents

TECHNICAL OVERVIEW · ARCHITECTURE · VISION

Third Edition — Expanded & Updated for the Infrastructure Baseline

IDENTITY IS A FUNCTION OF AUTHORIZATION AND ACCOUNTABILITY — NEVER REPUTATION.

Quantum Field Inc. · PRAXIS — The Trust Layer for the Agentic Economy
Reference implementation @praxis/agent-id · Tezos L1 + Arweave · Powered by Tezos · July 2026

Document Control

Title	PRAXIS Agent ID — Technical Overview, Architecture & Vision
Edition	**Third Edition** (expanded & updated). Supersedes the Second Edition; adds the engineering substrate (CI, drift gate, provenance releases), the golden byte-equality vectors, the ecosystem vision-to-code binding, and the phased infrastructure roadmap with Phase 0 complete.
Normative schema	`praxis:aid:1` · canonical encoding `praxis-canonical-json/1`
Reference implementation	`@praxis/agent-id` — TypeScript SDK, SmartPy FA2 contract, JSON Schemas, RDF ontology, golden vectors, CI pipeline
Substrate	Tezos L1 (soulbound FA2 anchor) + Arweave (permanent document storage)
Verification state	Clean strict typecheck · lint & format clean · **55/55 tests** (25 core + 17 KG + 13 golden-vector) · 4/4 examples schema-valid · chain + anchor re-verified · deliberate tamper rejected · **drift gate proven in both directions**
Knowledge-graph build	98 quads across 2 content-hash-named graphs · both pass `strictVerify` · graph commitment `27408e57…30d7a5f` · N-Quads proven in Oxigraph
Figure provenance	Every hash, quad count, test count, and commitment in this document is emitted by the package's own pipeline and is reproducible from a clean checkout via `npm ci && npm run ci` (the drift gate additionally proves the committed artifacts are exactly what the code generates).
Classification	Confidential — for institutional, regulatory, and standards-track review

Contents

Abstract

PRAXIS Agent ID defines a canonical, version-controlled, permanently stored identity record — the **Agentic Identity Document (AID)** — for a registered artificial-intelligence agent, committed on-chain by a **soulbound FA2 non-fungible token** on Tezos. The document lives on Arweave and is the source of truth; the token is the anchor that states, cheaply and verifiably, which identity version is authoritative right now, who controls it, and whether it remains live. A deterministic knowledge-graph projection turns those same anchored documents into an RDF/SPARQL-queryable lens whose every triple is re-verifiable against the on-chain commitment.

The entire system exists to answer one question for a skeptical counterparty — a bank compliance desk, a regulator, a settlement contract, another agent — without requiring trust in any intermediary: **who is accountable for this AI agent, what is it registered to do, which version of its identity is authoritative, at what assurance level, and is it still active?**

This Third Edition specifies what is being built and why — the design principles and invariants; the threat model and the attacks each mechanism defeats; the normative wire formats and the two non-circular hash rules; the three-layer anchoring architecture and its binding verification rule; the soulbound control model with two-step controller handoff; the smart-contract surface; the verifiable knowledge-graph layer; the codebase and its verification pipeline — and adds what the Second Edition predated: the **engineering substrate** that makes the repository infrastructure-grade (continuous integration across Node versions, an artifact-drift gate proven in both directions, provenance-attested releases, and governance files with a ranked threat model); the **golden byte-equality vectors** that let a second implementation prove conformance without reading TypeScript; the formal **vision-to-code binding** that maps every box of the Quantum Field ecosystem architecture to its implementing module or roadmap phase, with three doctrinal reconciliations; and the **phased infrastructure roadmap**, with Phase 0 complete.

Part I — What We Are Building

1.1 The problem, stated precisely

Autonomous AI agents are beginning to transact: initiating payments, executing settlement legs, negotiating with other agents, and operating inside regulated financial workflows. Every such action raises the same unresolved predicate questions: *on whose legal authority does this software act; how would a counterparty verify that authority without trusting the operator's own API; how is the authority's history — grants, amendments, revocations — proven rather than asserted; and who answers for the action in a dispute, after the fact, against adversarial incentives?*

Existing identity mechanisms fail these questions in characteristic ways. The failures are structural, not incidental, and they motivate every design decision that follows.

Mechanism	What it establishes	Why it fails the agent question
API keys / OAuth	A connection is authenticated	Authenticates a session, not an accountable principal; leaves no tamper-evident history a third party can replay.
PKI certificates	A key is bound to a name	Says nothing about delegated scope, assurance level, or lifecycle discipline; revocation is a weak, poll-based afterthought.
Reputation / scores	An aggregate signal of past behavior	Actively harmful here: farmable, purchasable, Sybil-inflatable, and uncitable by a regulator or court.
Web2 agent registries	A directory entry exists	Reduces to “trust the registry”; the registry is a single point of failure, capture, and silent edit.

PRAXIS Agent ID is engineered as the missing primitive: a **cryptographically anchored, append-only, standards-graded identity record** whose every claim is independently checkable from public artifacts alone — the on-chain anchor plus the permanently stored documents — with no reliance on the issuer's continued cooperation, honesty, or existence.

1.2 The artifact

Concretely, the system comprises seven interlocking deliverables, all present in the canonical repository and all verified by a single pipeline (`npm run ci`):

- **The AID document model and JSON Schemas** (`praxis:aid:1`) — the normative wire format, including standards-graded blocks for provenance, delegation, lifecycle, assurance, and security posture.
- **A TypeScript SDK** — canonical JSON hashing, tz4/base58check identity primitives, document construction and validation, version-chain verification, on-chain-anchor verification, point-in-time status computation, deterministic evidentiary rendering, and an NCCoE/OWASP compliance evaluator.
- **A SmartPy FA2 contract** (pre-compilation, audit-gated) — the soulbound Agent ID NFT with a two-step, acceptance-gated controller handoff and a complete TAP-8 event taxonomy.

- **A verifiable knowledge-graph layer** — a pure projection of anchored documents into RDF named graphs keyed by content hash, with containment-based verification, a hashable graph commitment, and a formal ontology.
- **Golden byte-equality vectors** (`fixtures/canonical-vectors.json`, encoding `praxis-canonical-json/1`) — acceptance and rejection vectors that let a second implementation prove hash conformance without reading TypeScript.
- **An engineering substrate** — CI across Node 20/22 with an artifact-drift gate, strict lint/format, and tag-driven releases with npm provenance attestation, so releasing from a drifted tree is impossible by construction.
- **Governance documents** — an agent-orientation brief with eleven non-negotiable invariants, a reconciliation record preventing silent re-divergence, a contribution contract, a security policy with ranked threat surface, and the vision-to-code binding (`ARCHITECTURE.md`).

1.3 The one-sentence design creed

Everything in the system derives from a single creed: **identity is a function of authorization and accountability, never reputation.** There is no score, no endorsement count, no activity volume anywhere in the schema — and, by standing invariant, there never will be. An identity that cannot show its accountable principal, its authorizing attestation, the assurance level of its keys, and the unbroken tamper-evident history of its own versions has not anchored identity; it has asked to be trusted. This package is built so it never has to ask.

1.4 Who this is built for

The design target is not a hobbyist deploying a bot; it is an institution that will be asked, by a regulator or a court, to account for what its software did. Each intended reader consumes a specific, purpose-built artifact:

Reader	What they need	What the package gives them
Bank compliance desk	A gateable, point-in-time determination of authority and status	`effectiveStatus` + the evidentiary rendering + validator-checked assurance floors
Regulator / examiner	An artifact they can cite, not a vendor claim	A deterministic evidentiary render and an honest compliance report naming both satisfied properties and gaps
Settlement contract / agent	A machine-checkable trust decision at speed	Re-hash against the anchor; walk the chain; require AAL3 + HSM + delegation-rooted
Auditor	Replayable history, not a snapshot	The append-only version chain and the TAP-8 transparency-log ethic
Court	Attribution to an accountable legal person	A delegation chain that must terminate in a depth-0 accountable root
Second implementer	Proof of conformance without reading this codebase	The golden vectors: reproduce every acceptance vector byte-for-byte; refuse every rejection vector

Part II — Design Principles & Objectives

2.1 Governing objectives

- **Independent verifiability.** Every claim must be checkable by a third party holding only the public artifacts. If a capability can only be verified by trusting the issuer or an indexer, it is designed wrong.
- **Legal legibility.** Outputs must be consumable by non-cryptographers exercising legal judgment: a deterministic evidentiary rendering for an examiner or court, and an honest compliance report naming both satisfied properties and gaps.
- **Standards grade.** Alignment with W3C DID/VC, NIST SP 800-63, the NIST NCCoE agent-identity properties, and the OWASP Non-Human-Identity Top 10 — enforced by validators, not asserted in prose.
- **Determinism.** Identical inputs produce byte-identical hashed artifacts across every implementation. No wall-clock reads inside identity logic, no float formatting in canonical output, no iteration-order dependence — and, since the Third Edition baseline, a published vector suite that makes the property externally testable.
- **Fail-closed defaults.** Unknown schema versions, unverified signatures, unconfigured providers, malformed inputs — all reject. A not-yet-wired capability throws; it never pretends to succeed.

2.2 The eleven invariants (non-negotiable)

The repository's orientation brief (`AGENT_BRIEF.txt`) fixes eleven invariants that bound all future evolution. A change that violates one is wrong even if every test passes. Condensed normative form:

I1	**No reputation.** No score, endorsement, ranking, or activity-volume field, ever.
I2	**The document is the source of truth**; the token holds only a hash commitment plus cheap discovery fields. Authoritative identity data never moves on-chain.
I3	**Indexers are never authoritative.** Anything fetched off-chain is re-hashed against the on-chain commitment before reliance (`verifyAgainstAnchor`).
I4	**Two hash rules, both non-circular.** A verifier checks both the per-version chain links and the constant genesis root — defense in depth.
I5	**Append-only, never delete.** Versions supersede; revocation is a terminal status flip; the chain and token are retained (the TAP-8 transparency-log ethic).
I6	**No hidden clocks.** Every temporal decision takes an explicit instant parameter (`effectiveStatus(doc, at)`).
I7	**Fail closed.** The default is deny; unwired capabilities throw.
I8	**Canonicalization is sacred.** Byte-exact canonical JSON — recursively sorted keys, no floats, no `undefined`, UTF-8, SHA-256 — identical in every implementation, now fixed externally by the golden vectors.
I9	**Soulbound + controller-vested + two-step handoff** is the control model. The NFT is never transferable; token custody never implies control.
I10	**Standards alignment is enforced, not decorative.** Every cited control is backed by a validator or compliance-evaluator check.

I11	**Honesty about boundaries.** Stubs, mocks, and unaudited components are stated plainly; a mock is never presented as real.

2.3 Trust-model doctrine: the binding rule

No single layer of the architecture is trusted alone. The NFT could point anywhere, so a resolver **re-hashes** the fetched Arweave document and compares it to the on-chain `aid_hash`. The Arweave document could claim any predecessor, so the version chain is verified **link by link back to genesis**, including the constancy of the genesis root. A resolver that checks only one of these has not verified identity. The knowledge-graph layer extends the same doctrine: a SPARQL result is advisory until each contributing named graph is re-verified by set containment against a re-projection of its anchored document.

2.4 Threat model — what each mechanism defeats

The doctrine above is best understood adversarially. For each realistic attack, the mechanism that defeats it and where that defense lives. The Third Edition extends the table to the repository itself — artifact tampering and release supply-chain attacks are now mechanically defended.

Attack	Defeated by	Enforcement point
Forged anchor — the token points at a document the agent never authored	Re-hash the fetched document; require equality with the committed hash	`verifyAgainstAnchor`
Indexer tampering — an off-chain cache serves an altered document	Indexers are never authoritative; every fetched byte is re-hashed	`aidContentHash + I3`
History rewriting — a version claims a false predecessor	Chain walked to genesis; each link's hash recomputed; genesis-root constancy checked	`verifyChain`
Content tampering at genesis	The immutable genesis root is an independent tripwire (second hash rule)	`genesisHash check`
Controller theft via token sale	Soulbound token; authority vested in the recorded controller, not custody	`FA2_TX_DENIED_SOULBOUND`
Silent controller change	Two-step, acceptance-gated handoff; every move is a logged event	`propose/accept_controller`
Assurance downgrade in the query layer (aal3 → aal1)	Containment verification against re-projection rejects the forged quad	`strictVerify`
Forged capability under a genuine graph label	A quad is admitted only if it is a member of the canonical projection	`verifyThenTrust`
Stale mandate — an identity outlives its authority	First-class expiry / rotation / offboarding; offboarded+active rejected	`effectiveStatus, validateAid`
Replay of an old version as current	The anchor names exactly one authoritative version and hash	`aid_hash + version`
Divergent second implementation — two honest parties disagree on bytes	Published acceptance + rejection vectors fix the encoding externally	`fixtures/canonical-vectors.json`

Attack	Defeated by	Enforcement point
Hand-edited “generated” artifact — examples silently diverge from code	CI regenerates everything and fails on any diff (proven in both directions)	`drift gate (.github/workflows/ci.yml)`
Release supply-chain — a package published from a tampered tree	Tag-gated publish runs the full gauntlet + drift gate; provenance attestation binds the tarball to the exact commit and workflow run	`release.yml, npm --provenance`

Part III — Technical Specification

3.1 Canonical JSON and record hashing (the root of determinism)

All hashing is defined over a closed, total canonical encoding (`praxis-canonical-json/1`) implemented in `src/codec/canonical.ts` with zero third-party dependencies. The rules: object keys sort recursively and lexicographically; `undefined` members are elided; strings are JSON-escaped UTF-8; booleans and `null` are literal; and **numbers must be finite safe integers** — floats, `NaN`, `Infinity`, and unsafe integers are rejected rather than coerced, so a hash can never depend on floating-point formatting. Monetary and other precision-sensitive protocol values are integers or decimal strings by rule. The record hash is `SHA-256(UTF-8(canonical(value)))`.

Every implementation — SDK, indexer, issuing wallet, contract-side verifier — must produce identical bytes (invariant I8). This is the foundation on which every other guarantee rests: if two honest parties can disagree about the bytes, they can disagree about the hash, and the anchor ceases to bind. The encoder is deliberately small enough to re-implement — and, since the Third Edition baseline, conformance is externally testable (§3.9).

3.2 Identity primitives: tz4 and did:tz

A PRAXIS agent identity **must** be a Tezos tz4 account — the BLS12-381 signature scheme — because BLS signatures aggregate: principal, agent, and witness signatures can combine into a single verifiable artifact rather than separate checks (TAP-1). `src/tap1/identity.ts` implements Base58Check from first principles (Bitcoin alphabet; 3-byte version prefixes [6,161,159] through [6,161,166] for tz1–tz4; checksum = first four bytes of double-SHA-256) and classifies addresses byte-for-byte, rejecting any corruption at the checksum. The stable cross-version identifier is the W3C-style DID `did:tz:<network>:<tz4>`, binding the account to a named network (`mainnet` | `ghostnet`). The DID is invariant across the identity's entire version history; the version chain moves beneath a fixed name.

3.3 The AidDocument: normative field model

The wire format is fixed by `schemas/aid.schema.json` (JSON Schema 2020-12, `additionalProperties: false`) with the discriminator `"schema": "praxis:aid:1"` — frozen per major version so verifiers fail closed on the unknown. The field groups:

Field group	Contents and constraints
Core identity	`schema`, `aid` (did:tz), `version` (monotonic, genesis = 1), `status` (`active` | `superseded` | `revoked`), `genesisHash`, `previousVersion { version, contentHash, arweaveTx }` (null only at genesis), `created` (constant), `updated`.
Accountability	`controller { address, display_name, legal_entity_ref }` — the accountable legal principal; `delegation[]` — a chain that **must** contain a depth-0 accountable root (validator-enforced; the anti-“unaccountable void” rule).
Authority surface	`agent` (tz4), `custodyClass` (`self` | `principal` | `threshold`), `capabilities[]` (dotted namespaces, e.g. `payments.transfer`) — a registry/discovery surface, **not** a grant; `cpoaAnchors[]` — SHA-256 pointers to live TAP-3 CPoA authority records.

Field group	Contents and constraints
Cryptography	`publicKeys[]` (typed `bls12_381` | `ed25519` | `p256` | `secp256k1`; purpose-tagged; unique ids enforced); `proof` — a detached signature envelope excluded from both hashes.
Assurance & risk	`assurance { ial, aal, fal, keyCustody }` (NIST SP 800-63 + `hsm`|`tee`|`mpc`|`software`); `provenance { model, modelVersion, operator, systemPromptHash, buildAttestation, workloadId }`; `security { promptInjectionControls, continuousAuthorization, leastPrivilege, humanApprovalTier, killSwitch }`; `lifecycle { expires, rotationPeriodSeconds, lastRotated, statusList, offboardedAt, offboardReason }`.
Interop & binding	`services[]` (DID-core endpoints); `credentials[]` (VC references by 64-hex hash + URI — **never inline PII**, hash mandatory); `adapters[]` (`native` | `mcp` | `a2a` | `ap2`); `registry { contract (KT1), tokenId, network }` — the bidirectional binding to the anchoring NFT.
Presentation	`display { name, description, image, tags }` — non-authoritative; an indexer that alters it fails the hash check.

3.4 Structural validation beyond shape

`validateAid` enforces cross-field semantics a schema alone cannot express: the did:tz address must equal the `agent` field; version 1 must have `previousVersion = null` and a self-consistent `genesisHash`; delegation depth must be a well-ordered chain with a depth-0 root; an offboarded identity must not claim `active` status; hash-shaped fields must be 64-hex; and capability namespaces must match `^[a-z0-9_-]+(\.[a-z0-9_-]+)*$`. Validation is total: an input either conforms or is rejected with a specific error — there is no partial acceptance.

3.5 The two hash rules (both non-circular)

```
contentHash(doc) = SHA-256( canonical( doc \ {proof} ) )

genesisHash      = SHA-256( canonical( genesis \ {proof, genesisHash} ) )
```

`contentHash` is the per-version hash: it anchors both the version-chain links and the NFT's `aid_hash`. Excluding `proof` resolves the signature fixpoint — a signer signs exactly the canonical unsigned bytes (`aidSigningBytes`) and attaches the envelope without disturbing the hash. `genesisHash` is the **immutable identity root**: derived once at genesis and carried unchanged through every later version. Excluding `genesisHash` from its own preimage is what makes the value well-defined. The payoff is defense in depth: content tampering is caught at the genesis-root tripwire even before the per-link check runs (I4).

3.6 Version-chain semantics

The chain is append-only (I5). `nextVersion` links backward by `(version, contentHash, arweaveTx)`, carries the genesis root forward, and **refuses to supersede a revoked identity** — revocation is terminal. `verifyChain` walks genesis-first and enforces, at every hop: structural validity; version increments of exactly one; constancy of `genesisHash` and of the `aid` identifier (identity-drift detection); and equality of each `previousVersion.contentHash` with the recomputed hash of the actual predecessor. Any break throws.

The verified real-hash chain shipped in `examples/` (regenerated on every build, never hand-edited — and drift-gated in CI):

`v1 contentHash`	`c2349d4a69dacea6d742a7371bfb1b314bd08ed98e2d06a0296b34d9e976b332`
`v1 genesisHash`	`1fe732078376b28a24aa09973d84aa876b64b18050740833663ecc22ce70394f`
`v2 contentHash`	`4ee2ffbc826fd90604fb0820c7409325ae49f2df316aa7a51626e17e2829eed4`
`invariants`	`v2.previousVersion.contentHash == v1.contentHash; genesisHash constant`

3.7 Token metadata, resolution, and point-in-time status

`toAgentIdTokenMetadata(doc, arweaveTx)` emits TZIP-12/16/21 metadata: `decimals: 0`, `isBooleanAmount: true`, `artifactUri: ar://<tx>`, and a `praxis` block carrying `{ standard, aid, agent, aid_hash, aid_uri, aid_version, genesis_hash, controller, soulbound: true, status }`. `verifyAgainstAnchor(fetched, anchor)` implements I3: re-hash the fetched document and require equality with `aid_hash`, plus genesis-root, version, identity, and controller consistency — the document is relied upon only if it is exactly what the NFT committed to. `effectiveStatus(doc, at)` computes status at an explicit instant with no hidden clock: revocation and offboarding dominate; an elapsed `lifecycle.expires` downgrades an otherwise-active identity to `expired`.

3.8 Evidentiary rendering and compliance evaluation

`renderEvidentiary(doc, at)` produces a deterministic, human-readable, court-and-examiner-facing text — identifier, accountable controller with legal reference, declared versus effective status at the stated instant, capabilities (marked declarative and CPoA-gated), the delegation chain to its accountable root, SP 800-63 assurance floors, and signature state. Same document and instant, byte-identical output. `evaluateCompliance(doc)` scores the document against the five NIST NCCoE identity-and-authorization properties and reports which OWASP NHI Top-10 risks the declared fields mitigate — **descriptively, never as certification**, with unsatisfied properties surfaced in an explicit `gaps` array. Honesty is a feature: the evaluator names what is missing.

3.9 Golden byte-equality vectors (new in this edition)

Invariant I8 is now externally testable. `fixtures/canonical-vectors.json` publishes **nine acceptance vectors** — each fixing `{ input, canonical, sha256 }` across recursive key sorting, unicode and escape handling, safe-integer bounds, deep nesting, and an AID-shaped record — and **three rejection vectors** an encoder must refuse (floats, negative floats, and 2^{53}, the smallest exactly-representable unsafe integer). A conforming second implementation — a Python indexer, a Rust verifier, contract-side tooling — reproduces every acceptance vector byte-for-byte and rejects every rejection vector, proving hash compatibility **without reading a line of this codebase**. The fixture is generated by the pipeline, replayed against the live encoder on every test run (13 dedicated tests), and covered by the CI drift gate — fixture and code cannot diverge silently.

Part IV — Architecture

4.1 The three-layer anchoring model

```
LAYER 1 — ON-CHAIN ANCHOR   Tezos FA2 (TZIP-12/16/21), soulbound NFT
  token_id <-> agent, 1:1, permanent. token_metadata commits to the
  CURRENT document via { aid_hash, aid_uri, version, genesis_hash,
  controller, status }. Answers: what is authoritative now, who
  controls it, is it live. Holds a COMMITMENT, never identity data.
                    | commits by SHA-256 to
                    v
LAYER 2 — PERMANENT STORAGE   Arweave (ar://<txid>)
  Every version written once, immutably, forever. The document is
  the SOURCE OF TRUTH; the token merely points and commits.
                    | each version hash-links to
                    v
LAYER 3 — VERSION CHAIN   v1 <- v2 <- v3 ...  (append-only)
  Each version names its predecessor by (contentHash, arweaveTx);
  genesisHash is the immutable identity root. A per-agent
  transparency log: never rewritten, never deleted.
```

The separation is load-bearing. Layer 1 buys liveness, control, and cheap on-chain discovery; Layer 2 buys survivability independent of any operator's server; Layer 3 buys auditable evolution and forgery detection. The binding rule (§2.3) welds them: no layer is trusted alone.

4.2 The control model: soulbound, controller-vested, two-step handoff

The ID-NFT is **soulbound**. An identity anchor that can be sold is a footgun — the buyer would appear to control the agent. The FA2 `transfer` and `update_operators` entrypoints exist so wallets and indexers recognize the contract, but they fail closed (`FA2_TX_DENIED_SOULBOUND`). Authority is vested in the **controller recorded in the anchor**, not in token custody; only the controller (or the TAP-5 governance multisig) may update or revoke. Because "who controls this identity" must always have exactly one unambiguous answer, changing the controller is a deliberate **two-step handoff**: `propose_controller` by the incumbent, then `accept_controller` by the successor. Control never moves without the incoming party's own on-chain consent, and every movement is a logged, discrete event — never a silent token transfer. A narrow, loudly-emitted administrative recovery path exists solely for institutional key loss and is gated behind governance before mainnet.

4.3 Contract surface (SmartPy, pre-compilation, audit-gated)

Entry point / view	Semantics
`anchor_identity`	Mints the Agent ID NFT; enforces 1:1 agent ↔ token, 32-byte hashes, controller-or-governance authorization; sets the immutable `genesis_hash`. Emits `registration`.
`update_identity`	Moves the anchor to a new AID version; `genesis_hash` never touched; version strictly increases; revoked identities cannot be updated. Emits `metadata_update`.
`revoke_identity`	Irreversible offboarding (OWASP NHI1); record and token retained

Entry point / view	Semantics
	forever. Emits `identity_revoked`.
`propose_controller / accept_controller`	The two-step handoff; control moves only on acceptance. Emits `controller_handoff_proposed` / `_accepted`.
`transfer / update_operators / balance_of`	Full FA2 required interface: transfers and operator updates fail closed (soulbound); `balance_of` answers 0/1 per TZIP-12.
`resolve / token_id_of_agent / get_status`	On-chain views. `resolve` returns the anchor so a client can fetch `ar://aid_uri` and re-hash — a convenience only; the client must still re-verify (I3).

Contract invariants: token ↔ agent is 1:1 and permanent; `genesis_hash` is write-once; every state change emits a TAP-8 event (no silent mutation); revocation is terminal; only controller or governance mutates. The five event tags are registered into the TAP-8 taxonomy — the add-to-taxonomy-before-ship discipline.

4.4 The Arweave boundary (fail-closed by construction)

The SDK deliberately ships **no live Arweave client**. `ArweavePublisher` and `ArweaveResolver` are interfaces; the shipped defaults (`Unconfigured*`) throw `ArweaveNotConfiguredError` rather than fabricating a transaction id or empty bytes. `aidUploadBytes(doc)` fixes the exact bytes every publisher must upload — the canonical serialization — so the on-chain hash always matches what a resolver recomputes; `aidUploadTags` standardizes gateway-discoverable tags. Wiring a real bundler (arweave-js, Irys/Turbo, a KMS-signed uploader) is a Phase 2 integration event that must preserve the interface and the fail-closed default.

4.5 The verifiable knowledge-graph layer

The knowledge graph is a **lens, never a source of truth** — invariants I2/I3 lifted to the graph layer. `documentToQuads` is a pure, deterministic projection of a validated AID into RDF quads, every one emitted into a **named graph whose IRI embeds the asserting version's content hash**: `urn:praxis:sha256:<aidContentHash>`. Since that is the very hash the NFT anchors, *graph provenance is the on-chain commitment* — a SPARQL result returns, in `?g`, the anchored hash behind every answer.

Verification is **set containment against re-projection**, deliberately stronger than hash checking. A graph label only names a document; it does not enumerate quads, so a malicious indexer could attach forged triples under a perfectly genuine label. `verifyThenTrust` therefore: (1) parses the graph label; (2) obtains the anchored document from a resolver that must supply only anchor-verified documents; (3) requires the document to re-hash to the label; and (4) **re-projects** it and admits a presented quad only if it is a member of the canonical projection. Rejections carry reasons and are never silent; `strictVerify` fails closed on the first. What the KG test suite proves: a forged capability under a valid label is rejected; a one-byte assurance downgrade (`aal3` → `aal1`) is rejected; unknown graphs are rejected rather than skipped; a lying resolver is caught; and honest subsets — which is exactly what query results are — verify cleanly.

Vocabulary reuses before inventing: identities are did:tz IRIs; lineage and delegation ride W3C PROV-O; capabilities are **shared IRIs** under `w3id.org/praxis/capability#`, so two operators' `payments.transfer` meet at one node — the semantic-interop payoff. PRAXIS-specific terms live in

`ontology/praxis.ttl`, each annotated with the enforced control it reflects (I10). **No blank nodes** are emitted (delegation links receive deterministic IRIs), keeping the sorted N-Quads byte-stable and the containment check exact; `quadsCommitment()` therefore yields one SHA-256 for an entire graph — a value a contract, CPoA record, or TAP-8 log entry can commit to. Because verification is per-graph and set-union-safe, independently built indexes **federate by merging quads**, each side re-verifying for itself. The shipped artifacts (98 quads across 2 named graphs, commitment `27408e57…30d7a5f`) are proven against a real engine: they load into Oxigraph and the three worked queries (discovery, lineage, interop) return the expected rows.

Part V — Codebase Structure & Verification Pipeline

5.1 Repository layout

```
praxis-agent-id/
  AGENT_BRIEF.txt            agent orientation: invariants, package map, definition of
done
  RECONCILIATION.md          how the package became canonical; merge provenance
  ARCHITECTURE.md            vision-to-code binding (Part VII of this document)
  CONTRIBUTING.md / SECURITY.md / LICENSE
  .github/workflows/         ci.yml (gauntlet + drift gate, Node 20/22) · release.yml
(provenance)
  schemas/                   NORMATIVE: aid.schema.json · aid-fa2-token-
metadata.schema.json
  src/
    codec/canonical.ts       canonical JSON + SHA-256 (zero-dependency; root of I8)
    tap1/identity.ts         base58check, tz1-tz4 classification, tz4 agent rule
    aid/aid.ts               THE CORE: document, hashing, chain, anchor, resolution,
                             effectiveStatus, renderEvidentiary, evaluateCompliance
    aid/arweave.ts           fail-closed publish/resolve boundary
    kg/kg.ts                 KG projection, N-Quads, commitment, containment verify
  contracts/agent_id_nft.py  SmartPy soulbound FA2 (pre-compilation, audit-gated) +
DESIGN.md
  ontology/praxis.ttl        the PRAXIS ontology (every term backed by a check)
  fixtures/canonical-vectors.json   golden byte-equality vectors (praxis-canonical-
json/1)
  scripts/                   build_examples.mts · build_kg.mts · build_vectors.mts ·
validate.mjs
  examples/                  generated, never hand-edited: documents, token metadata,
                             compliance report, evidentiary renders, kg/ (N-Quads,
                             commitment, context), sparql/ (3 worked queries)
  test/                      aid.test.ts (25) · kg.test.ts (17) · vectors.test.ts (13)
  docs/                      AID_DESIGN.md · KG_DESIGN.md · this overview · reference
card
```

5.2 Toolchain discipline

TypeScript in maximal strict mode — `strict`, `noUncheckedIndexedAccess`, `exactOptionalPropertyTypes`, `noImplicitOverride`, `verbatimModuleSyntax` — with ESM (`NodeNext`) resolution, now including the build scripts themselves (an escape the Third Edition baseline closed). Strict ESLint (type-aware, `no-explicit-any`, `consistent-type-imports`) and Prettier are enforced in CI. Runtime dependencies: **none**; dev dependencies only. The SDK ships no Arweave client, no triplestore, and no SPARQL engine by design: boundaries are interfaces, integrations are explicit events.

5.3 The verification pipeline

```
npm run ci
  = typecheck (strict)  ->  lint  ->  format check
  -> build:examples     regenerate every example from the SDK (real hashes)
  -> build:kg           project the chain to N-Quads; strictVerify round-trip
  -> build:vectors      regenerate the golden byte-equality vectors
  -> validate           AJV (2020-12) + verifyChain + verifyAgainstAnchor
```

```
                      + a deliberate tamper that MUST be rejected
-> test               55 tests (25 core + 17 KG + 13 vectors)
-> drift gate         git diff --exit-code -- examples fixtures
```

Current verified state, from a clean install: strict typecheck clean; lint and format clean; **55/55 tests passing**; all four examples schema-valid; the two-version chain verifies with constant genesis root; both documents match their on-chain anchors; the injected tamper is rejected; the knowledge-graph build emits 98 quads across 2 named graphs, both passing `strictVerify`; and the golden vectors replay exactly. Examples and fixtures are generated artifacts — regenerating them is part of the pipeline precisely so schemas, SDK, and artifacts can never drift apart unnoticed.

5.4 The engineering substrate (Phase 0, complete)

The Third Edition baseline adds the substrate that makes the repository infrastructure-grade rather than merely correct:

- **Continuous integration** across Node 20 and 22 on every push and pull request, running the entire gauntlet above.
- **The artifact-drift gate.** CI regenerates `examples/` and `fixtures/` from source and fails on any git diff — the mechanical enforcement of "generated artifacts are never hand-edited." The gate is **proven in both directions**: a committed hand-edit fails CI (exit 1); the honest tree passes (exit 0).
- **Provenance-attested releases.** Publishing is tag-driven, requires the tag to match the package version, runs the full gauntlet plus drift gate first, and publishes with npm provenance — cryptographically binding the tarball to the exact commit and workflow run. Releasing from a drifted tree is impossible by construction.
- **Governance files.** `CONTRIBUTING.md` (the contribution contract, deferring to the brief's invariants; every new rule ships with a positive **and** a negative test), `SECURITY.md` (private disclosure channel plus a **ranked threat surface** — the canonical encoding at the top, the unaudited contract explicitly scoped), and an MIT `LICENSE`.

5.5 How a counterparty verifies an agent — end to end

The doctrine reduces to a short, mechanical procedure any third party can run with only public artifacts. No step trusts the issuer.

- **Read the anchor.** Call the contract's `resolve` view (or read token metadata) to obtain `{ aid_hash, aid_uri, version, genesis_hash, controller, status }`.
- **Fetch the document.** Retrieve the AID from `ar://aid_uri` — from Arweave directly, or any gateway; the source is untrusted.
- **Re-hash and bind.** Recompute `aidContentHash` and require equality with `aid_hash` (`verifyAgainstAnchor`). If it differs, stop — the anchor does not bind this document.
- **Walk the chain.** Fetch each predecessor by `previousVersion` back to genesis; `verifyChain` confirms the links and the constant genesis root.
- **Compute point-in-time status.** Evaluate `effectiveStatus(doc, at)` for the instant that matters to the decision — no hidden clock.
- **Gate on policy.** Require the delegation root, the assurance floor (e.g. AAL3 + HSM), and any capability/CPoA condition the transaction demands.

- **Optionally query the graph.** Any discovery answer is advisory until each contributing named graph passes `strictVerify` against a re-projection — the query layer can never launder a forged claim.

Part VI — Standards Alignment (Enforced, Not Decorative)

The discipline of invariant I10 governs this entire part: where the package claims a standards property, a validator or the compliance evaluator actually checks it. Citing a control without wiring its check is treated as a defect. The right-hand column names the enforcement point, not an intention.

Standard / framework	How alignment is realized and enforced
W3C DID / Verifiable Credentials	`aid` is a did:tz; `publicKeys`/`services`/`credentials` follow DID-core shapes; credentials are hash/URI references with off-document selective disclosure — inline PII is structurally impossible (hash mandatory, enforced).
NIST SP 800-63	`assurance` carries IAL/AAL/FAL plus key custody so a relying party can require a floor; enum domains are validator-enforced; the evidentiary rendering surfaces the floors verbatim.
NIST NCCoE (software & AI-agent identity and authorization)	`provenance`, `delegation`, and the audit chain map to identification, non-repudiation, and auditing; `evaluateCompliance` scores all five NCCoE properties and reports gaps honestly. The delegation chain must reach a depth-0 accountable root — enforced, not suggested.
OWASP Non-Human-Identity Top 10	`lifecycle`/`assurance`/`security` actively mitigate NHI1 (offboarded+active rejected), NHI2 (hardware custody), NHI5 (least-privilege/continuous authorization), NHI7 (expiry/rotation), NHI9 (workload identity), NHI10 (the delegation root). The evaluator reports which risks a document actually mitigates.
Tezos TZIPs	TZIP-12 (FA2 interface, soulbound-constrained), TZIP-16 (contract metadata), TZIP-21 (rich token metadata), with an explicit FA2.1/TZIP-26 migration note for native events, views, and transfer policies.
W3C PROV-O / RDF / SPARQL / JSON-LD	Lineage and delegation reuse PROV-O terms; the graph ships a JSON-LD context and a Turtle ontology; determinism is achieved without blank nodes, with W3C RDC-1.0 named as the upgrade path should blank nodes ever be introduced.
PRAXIS TAPs	TAP-1 (tz4 identity), TAP-2 (non-authoritative registry resolution), TAP-3 (CPoA live authority — capabilities here are discovery, never grant), TAP-5 (governance multisig), TAP-8 (append-only transparency; the five contract events are registered into the taxonomy).

Part VII — Ecosystem Binding: Vision to Code

The Quantum Field ecosystem architecture — *The Semantic Trust Infrastructure for the Agentic Economy* — describes the full stack: AI agents reasoning inside a trust layer, a knowledge graph for meaning, a legal ontology and policy engine for institutional rules, and Tezos as the cryptographic anchor. The repository's `ARCHITECTURE.md` formally binds every box of that diagram to this codebase, with a four-value status vocabulary kept honest under I11: **SHIPPED** (enforced by `npm run ci` today), **PARTIAL** (the seam exists here; the remainder has a named phase), **ROADMAP(N)** (a phase of Part VIII's plan), and **VISION** (a sibling system, out of this repository's scope by design). Highlights:

Vision boxes	Status	Where / why
Agent Identity · Provenance Records · Trust Assertions · Knowledge Graph · Evidence Records · Governance Events	**SHIPPED**	The AID + soulbound anchor; the append-only chain; `evaluateCompliance`/`renderEvidentiary`/`effectiveStatus`; `src/kg/`; the version chain + TAP-8 events.
Credential Management · Authority Delegation · Verification APIs · Governance Controls · Authorization Proofs · Legal Ontology	**PARTIAL**	Hash-only credential refs; delegation chains + `cpoaAnchors` (live CPoA evaluation is TAP-3); verification *functions* shipped, hosted resolver + verifier CLI in Phase 3; two-step handoff shipped, multisig deployment in Phase 1; the `proof` envelope shipped, BLS verification in Phase 2; `praxis.ttl` as the identity stratum of the full institutional ontology.
Policy Engine · Reasoning Engine	**VISION**	Deliberately absent here: capabilities are discovery, never grant. The seams a policy engine consumes (capabilities, assurance floors, security posture, CPoA anchors) are shipped fields; rule evaluation belongs to TAP-3 / Symbolic Agent.

7.1 Doctrinal reconciliation — evidence accumulates; trust is recomputed

The ecosystem vision's trust loop ends “Knowledge Graph Updated → Future Decisions Improved.” Read as *evidence accumulates*, that is exactly this system. Read as *trust accrues*, it would violate invariant I1 — accrued trust can be farmed, purchased, or Sybil-inflated. The binding fixes the reading: **every decision is evaluated from scratch** against anchored identity, live authority, and policy at an explicit instant. There is no balance that grows, no score, no endorsement count — and by I1 there never will be. What improves over time is the *evidence base*, never a trust number. This is also the differentiator: reputation-based registries cannot make this statement.

7.2 Doctrinal reconciliation — the graph proposes, the anchors confirm

The vision has the knowledge graph *answering* authority queries. In this architecture that phrasing is precise only with its second half attached: the graph is how a verifier **discovers** the claimed authority chain in milliseconds; the claim is **confirmed** only by re-verification against the anchored AID (containment against re-projection) and, for live authority, the TAP-3 CPoA record. Without this rule the architecture would reintroduce “trust the registry” — the failure mode the vision itself criticizes. With it, a SPARQL answer arrives carrying, in `?g`, the exact on-chain commitment that proves it.

7.3 Naming — “Tezos X”

Outward material brands the anchor layer **Tezos X** (the platform's forward roadmap). The normative surfaces of this repository bind to what is verifiable today: Tezos L1 semantics, networks `mainnet` | `ghostnet`, FA2/TZIP-12/16/21, with an FA2.1 (TZIP-26) migration note in the contract — so an auditor always knows exactly which chain a KT1 lives on.

Part VIII — The Ultimate Vision & the Problems It Solves

8.1 The world this is built for

The next phase of the digital economy is agentic: software principals that hold mandates, move value, and contract with one another at machine speed. The binding constraint on that economy is not model capability — it is **accountability infrastructure**. A bank will not let an agent touch a settlement rail, a regulator will not bless an agent-mediated product, and a court cannot adjudicate an agent-caused loss unless the agent's authority is provable, its lineage inspectable, and its accountable principal identifiable — independently, after the fact, and against adversarial incentives. PRAXIS Agent ID is the identity layer of that infrastructure: the TAP-1/TAP-2 stratum of the wider PRAXIS protocol, designed to interlock with TAP-3 (the Cryptographic Power of Attorney — the live, scoped authority an agent acts under), TAP-5 (governance), and TAP-8 (the transparency-log ethic), and conceptually aligned with the CPoA framework submitted to the NIST NCCoE public process on AI-agent identity and authorization.

8.2 How the layers compose

Identity is necessary but not sufficient. The full trust decision an institution must make decomposes cleanly across the PRAXIS stack, and Agent ID is the load-bearing base:

Question	Layer	Answered by
What is this entity, and is this version authoritative?	Identity (TAP-1/2)	The AID + soulbound anchor (this document)
What, precisely, is it authorized to do right now?	Authority (TAP-3)	CPoA records referenced by `cpoaAnchors[]`
Who governs changes to it?	Governance (TAP-5)	The controller + governance multisig
What happened, and can it be replayed?	Transparency (TAP-8)	The append-only chain + emitted events

The distinction between `capabilities[]` and CPoA is doctrinally central: capabilities are a **discovery surface** (“this agent is registered to attempt payments”), never a grant. Actual authority to act is the live, scoped, revocable CPoA — default-deny at the point of action. Identity says *who* and *what*; authority says *whether, right now*.

8.3 The problems it solves, precisely

Problem	Resolution
Attribution	Every action traces to a did:tz identity whose delegation chain reaches a named, accountable, depth-0 legal principal — agency-law legibility by construction, not forensic reconstruction.
Verification without trust	Any third party, holding only public artifacts, can re-derive every claim: re-hash the document against the anchor, walk the chain to genesis, re-project the graph, replay the golden vectors. The issuer's cooperation is never required.

Problem	Resolution
Tamper-evident history	The append-only chain plus the constant genesis root make silent rewriting detectable; point-in-time status is computable for any instant — the substrate for audits, discovery, and dispute resolution.
Lifecycle discipline	Expiry, rotation, offboarding, and terminal revocation are first-class and enforced, so identities cannot silently outlive their mandate — the failure mode behind most non-human-identity incidents.
Assurance-floor commerce	Counterparties can gate on declared, validator-checked IAL/AAL/FAL and key custody — "only transact with agents at AAL3, hardware custody, delegation-rooted" becomes an executable policy.
Ecosystem-scale discovery	The verifiable knowledge graph makes the registry queryable — who can do what, under whom, since when — while containment verification guarantees the query layer can never launder a forged claim.

8.4 The infrastructure roadmap (Phase 0 complete)

The Second Edition's maturation arc is now a phased, dependency-ordered plan, each phase gated on the one before. The organizing principle: convert the package from *specification with labeled mocks* to *operable infrastructure with a live existence proof* — without ever eroding the fail-closed and honesty disciplines that make it credible.

`Phase 0`	**Engineering substrate — COMPLETE.** CI across Node 20/22; the artifact-drift gate (proven in both directions); strict lint/format; golden byte-equality vectors; provenance-attested releases; LICENSE / SECURITY.md / CONTRIBUTING.md.
`Phase 1`	**Contract: compile, test, audit, originate.** Machine-check the SmartPy; scenario tests for every authority path, positive and negative; independent audit scoped to anchor/update/revoke/handoff; Ghostnet origination with TZIP-16 metadata; TAP-5 multisig deployment and admin sunset.
`Phase 2`	**Real providers behind the existing interfaces.** An Irys/Turbo publisher implementing `ArweavePublisher` (ArLocal in CI); BLS12-381 verification — including aggregate — behind the proof envelope. Neither may loosen a fail-closed default. **Milestone: the existence proof** — one real end-to-end identity on Ghostnet: real TxID, real signature, real anchor, resolved and verified.
`Phase 3`	**The service layer.** A TAP-8-event-driven indexer feeding a hosted Oxigraph (only verified quads enter); a resolver API with point-in-time status; and a third-party **verifier CLI** (`praxis-aid verify did:tz:…`) anyone can run with zero trust in the operator.
`Phase 4`	**Standards hardening.** Register `w3id.org/praxis`; host the ontology, JSON-LD context, and versioned schemas at stable URLs; TAP-1/TAP-2 stable drafts; the NCCoE-facing paper, with containment verification presented under its own name.
`Phase 5`	**Security & operational maturity, before mainnet.** Written threat model across all layers; property-based fuzzing and differential testing of the canonicalizer; key-management and multisig-ceremony runbooks; revocation/incident runbooks; divergence monitoring; SBOM and supply-chain gates.

8.5 Honest current boundaries

Stated plainly, per invariant I11: the smart contract is an uncompiled, unaudited, pre-deployment draft written for review. Signature verification and the live Arweave client are gated behind their audited providers and fail closed until wired (Phase 2). Example Arweave transaction ids are deterministic mocks and example proofs are absent; every hash, chain link, and vector, by contrast, is real and reproducible from the pipeline — and drift-gated. The `w3id.org/praxis` identifiers await registration (Phase 4); the N-Quads serialization is deterministic but is not W3C RDC-1.0 (unnecessary absent blank nodes, and named as the upgrade path). `evaluateCompliance` describes; it does not certify.

8.6 Closing

The measure of every contribution to this system is fixed: does it make the identity more independently verifiable, more legally legible, and more standards-grade — without adding reputation, weakening a fail-closed default, or introducing a hidden clock or a silent mutation? An identity that cannot show its accountable principal, its authorizing attestation, the assurance of its keys, and the unbroken history of its own versions has not anchored identity — it has asked to be trusted. **This system is built so it never has to ask.**

Quantum Field Inc. · PRAXIS · The Future Will Be Verified.