

# AEP v2.5 - Agent Element Protocol - Specification Reference

Version: 2.5.0

Date: April 2026

Author: the.PM / New Lisbon Agency / IPHCCP

Licence: Apache 2.0

Repository: [github.com/thePM001/AEP-agent-element-protocol](https://github.com/thePM001/AEP-agent-element-protocol)

Website: [aep.newlisbon.agency](https://aep.newlisbon.agency)

Tests: 758 across 51 files

Features: 73 across 11 categories

---

## 1. ARCHITECTURE

Three independent layers:

Structure (aep-scene.json): topological IDs, hierarchy, spatial rules, z-bands

Behaviour (aep-registry.yaml): actions, states, constraints, forbidden patterns

Skin (aep-theme.yaml): colours, typography, design tokens

Changing one layer never requires changing another.

---

## 2. MATHEMATICAL FOUNDATION

Deterministic Adjudication Lattice (DAL):

Population of candidate outputs filtered through hierarchical verification predicates.

Convergence theorem: zero-defect selection with  $N = O(\ln(1/\delta)/\alpha_T)$  candidates.

Centroid absorber:  $O(1)$  drift bounds independent of sequence length.

Freshness predicate: prevents stale-context hallucinations.

Lattice Memory:

Append-only record of every validation decision.

Attractor system from accepted proposals.

Fast-path: proposals matching attractors above 95% similarity skip cold-path validation.

TLA+ invariant: `MemoryDoesNotAffectDecision`. Memory guides but never overrides.

---

## 3. 15-STEP EVALUATION CHAIN

Step 0: Task scope check (if decomposition active)

Step 1: Session state check

Step 2: Ring capability check

Step 3: System-wide rate limit

Step 4: Per-session rate limit

Step 5: Intent drift check (skip during warmup)

Step 6: Escalation check  
Step 7: Covenant evaluation (hard + soft severity)  
Step 8: Rego / forbidden pattern check  
Step 9: Capability match + trust tier check  
Step 10: Budget/limit check  
Step 11: Gate check (human or webhook)  
Step 12: Cross-agent verification (multi-agent only)  
Step 13: Knowledge retrieval validation (if knowledge base active)  
Step 14: Content scanner pipeline (11 scanners)

After step 14: hard violation rejects. Soft violation triggers recovery.  
All pass: approve, log tokens/cost, update trust, export OTEL span.

---

## 4. TRUST SCORING

Range: 0-1000

Five tiers:

Untrusted: 0-199

Provisional: 200-399

Standard: 400-599

Trusted: 600-799

Privileged: 800-1000

Trust changes:

Successful action: +5 to +20

Soft violation recovered: -10

Soft violation exhausted: -50

Hard violation: -10 to -100

Workflow advance: +15

Workflow rework: -20

Workflow skip: -5

Workflow fail: -100

Idle erosion: configurable rate per minute

---

## 5. EXECUTION RINGS

Ring 0 (Kernel): full access, all operations

Ring 1 (System): create, update, delete within policy

Ring 2 (User): create, update only

Ring 3 (Sandbox): read-only

Trust loss triggers automatic demotion.

Ring assignment per workflow phase.

---

## 6. BEHAVIOURAL COVENANTS

Three keywords: permit, forbid, require

Severity annotation: [hard] or [soft]

Examples:

```
permit aep:create (prefix in ["CP", "PN"]);
```

```
forbid aep:delete (prefix == "SH") [hard];
```

```
forbid file:write (path matches "*.tmp") [soft];
```

```
require trust_tier >= "standard" [hard];
```

```
permit commerce:add_to_cart (merchant in ["store_123"]);
```

```
permit knowledge:retrieve (scope in ["product_docs"]);
```

```
forbid knowledge:retrieve (scope == "internal_finance");
```

Covenants are agent-declared and portable.

Rego policies are operator-enforced and environment-specific.

---

## 7. HARD/SOFT VIOLATION DISTINCTION

Hard violations: immediate reject. No retry. Existing behaviour.

Soft violations: recovery attempt before rejection.

Recovery flow:

1. Soft violation detected (covenant, policy or scanner)
2. Build corrective prompt with violation details
3. Agent regenerates with feedback
4. Re-run through full 15-step chain
5. If still failing after k attempts (configurable, default 2):  
escalate to hard reject

Evidence ledger entries: recovery:attempt, recovery:success, recovery:exhausted

---

## 8. CONTENT SCANNER PIPELINE

11 built-in scanners. Runs after structural validation, before approval.

Scanner	Checks	Default Severity
PII	Email, phone, SSN, credit card, NIF	hard
Injection	SQL, XSS, SSTI, command injection	hard
Secrets	API keys, private keys, .env patterns	hard
Jailbreak	Prompt override, DAN mode, ignore rules	hard
Toxicity	Slurs, hate speech, threats	soft
URLs	URL detection with allowlist/blocklist	soft
Data Profiler	Null rates, duplicates, outliers, schema	soft

Prediction	Extreme percentages, false confidence	soft
Brand	Required phrases, competitors, tone	soft
Regulatory	Ad disclosure, financial/medical disclaimers	hard
Temporal	Stale dates, expired content, undated stats	soft

Integration with streaming: hard findings mid-stream abort immediately.

Integration with recovery: soft findings trigger recovery engine.

---

## 9. WORKFLOW PHASES

Sequential pipeline stages with typed verdicts.

PhaseVerdict types:

advance: move to next phase

rework: return to current phase with feedback

skip: bypass with justification logged

fail: terminate workflow or escalate

Each phase defines:

name, description, entry conditions, role, ring,

exit criteria, max rework count

Built-in template - Fine-tuning workflow (6 phases):

1. DATA\_PREPARATION - import and validate training data
  2. DATA\_VALIDATION - compute baseline metrics, check distribution
  3. TRAINING\_CONFIG - configure and validate hyperparameters
  4. TRAINING\_EXECUTION - run training under governance
  5. EVALUATION - compare against baseline, fail if degraded
  6. DEPLOYMENT - human gate approval, proof bundle generation
- 

## 10. MODEL GATEWAY

Provider-agnostic governed model calls.

4 built-in providers:

Anthropic (Claude) - Messages API with SSE streaming

OpenAI (GPT) - Chat Completions with SSE streaming

Ollama (local) - Chat API with NDJSON streaming

Custom - any OpenAI-compatible endpoint

All raw fetch(). Zero SDK dependencies.

Every call:

1. Inject governance context (prompt optimization)
2. Call model via provider adapter
3. Validate response through full 15-step chain

4. Hard violation: reject with details
5. Soft violation: trigger recovery engine
6. Record tokens and cost in evidence ledger
7. Export OTEL span (if enabled)

Streaming: AsyncGenerator with mid-stream abort on hard violations.

---

## 11. FLEET GOVERNANCE

Swarm-level policies across all active agents.

Fleet policies:

max\_agents: max simultaneous active agents

max\_total\_cost\_per\_hour: planetwide cost cap

max\_ring0\_agents: max kernel-ring agents simultaneously

drift\_pause\_threshold: N drifting agents triggers swarm pause

require\_parent\_covenant\_subset: child agents inherit parent scope

Spawn governance:

Child covenant must be subset of parent covenant

Child cannot permit anything parent forbids

Child cannot skip any parent require

Child ring  $\geq$  parent ring (same or lower privilege)

Child trust starts at parent trust \* 0.8

Message scanning:

Inter-agent messages pass through full scanner pipeline.

Poisoned instructions, PII leaks and injection between agents are blocked.

Fleet API:

GET /fleet/status, GET /fleet/agents, GET /fleet/alerts

POST /fleet/pause, POST /fleet/resume, POST /fleet/kill

---

## 12. KNOWLEDGE BASE

Governed RAG with lattice-level validation.

Ingestion:

Content split into chunks (configurable size, default 500 tokens)

Each chunk scanned by full scanner pipeline

Hard failure: rejected (not stored)

Soft failure: flagged (stored with warning)

Clean: validated (stored as approved)

Retrieval:

Covenant-scoped (agent only sees chunks within permitted scope)

Double-scanned (once at ingest, once at retrieval)  
Anti-context-rot ordering:  
Most relevant chunk at position 1 (context start)  
Second most relevant at position LAST (context end)  
Remaining in middle (least attention-critical)  
Counteracts U-shaped attention curve.

---

### **13. EVAL-TO-GUARDRAIL LIFECYCLE**

Production -> Dataset -> Eval -> Rules -> Policy

EvalRunner: runs datasets against governance policies.

RuleGenerator: analyses violation patterns, generates suggested:

- Covenant forbid rules for structural patterns that fail
- Scanner regex patterns for content that fails

Suggestions are human-reviewable. Never auto-adopted.

DatasetManager:

Create, version, import from production ledgers

Export to JSON and CSV

Actions allowed -> expectedOutcome: "pass"

Actions denied -> expectedOutcome: "fail"

---

### **14. PROMPT OPTIMIZATION**

Governance context injection:

Prepends permitted actions, forbidden actions, required conditions,  
trust tier, ring, scanner categories to agent prompts.

Guidance, not enforcement. Covenants and policies handle enforcement.

Eval-based refinement:

Analyses which violations occur most with a prompt.

Adds specific instructions to avoid top violation patterns.

Prompt comparison:

Runs two prompts against same dataset under same governance.

Reports which produces fewer violations and higher theta.

Prompt versioning:

Save, load, list, diff with SHA-256 hashes.

---

### **15. ML METRICS**

Classification: accuracy, precision, recall, F1, confusion matrix, support

Regression: MSE, RMSE, MAE, R2, MAPE

Retrieval: precision@k, recall@k, MRR, NDCG

Generation: exact match, average length, empty rate

Composite score feeds into reliability index.

---

## 16. COMMERCE SUBPROTOCOL

12 commerce actions: discover, add\_to\_cart, remove\_from\_cart, update\_cart, checkout\_start, checkout\_complete, payment\_negotiate, payment\_authorize, fulfillment\_query, order\_status, return\_initiate, refund\_request

Commerce policies:

max\_transaction\_amount, allowed\_currencies, allowed\_merchants, blocked\_merchants, blocked\_product\_categories, require\_human\_gate\_above, allowed\_payment\_methods, max\_daily\_spend

Spend tracking: daily accumulator with JSONL persistence.

Merchant registry: CRUD for merchant profiles.

Covenant extensions: commerce-specific permit/forbid/require rules.

---

## 17. SECURITY

SHA-256 hash chain: append-only evidence ledger, tamper one entry and the chain breaks at that sequence number

ML-DSA-65: post-quantum digital signatures (FIPS 204) per entry, opt-in

RFC 3161: standardized timestamp authority, async and batched

Merkle proofs: verify individual entries without full chain, computed every 100 entries (configurable)

Offline sync: queue signed actions locally, verify on reconnect

System rate limit: planetwide cap across all sessions

Proof bundles: signed portable session verification artifact containing identity, covenant, statistics, Merkle root, ledger hash, trust, ring, drift, reliability index, signature

Human gates: operator approval before execution

Webhook gates: automated HTTP callback approval

---

## 18. COMPLIANCE

OWASP Agentic Top 10: all 10 risks covered

AT-01 Goal hijacking: intent drift + covenants

AT-02 Tool misuse: policy + rings

AT-03 Identity abuse: agent identity + trust  
AT-04 Supply chain: covenant exchange  
AT-05 Code execution: rings + shell proxy  
AT-06 Memory poisoning: read-only ledger  
AT-07 Insecure comms: ML-DSA-65  
AT-08 Cascading failures: kill switch + rate limits  
AT-09 Human trust exploit: gates + tiers  
AT-10 Rogue agents: trust erosion + auto-demotion  
  
EU AI Act: high-risk obligation evidence collection  
SOC 2: Type II control documentation  
Report export: JSON, CSV, HTML

---

## 19. OBSERVABILITY

OpenTelemetry exporter:  
session:start -> root span  
action:evaluate -> child span (tool, decision, trust\_change, ring)  
recovery:attempt -> child span of action  
scanner findings -> span events  
workflow:phase\_enter -> child span of root

Token and cost tracking:  
Per-action: input tokens, output tokens, total tokens  
Per-action: input cost, output cost, total cost  
Per-session: total tokens, total cost, cost saved  
Cost saved estimated from early aborts and rejections

Reliability index (theta):  
hardComplianceRate: 0-1 (weight 0.3)  
softRecoveryRate: 0-1 (weight 0.2)  
driftScore: 0-1 (weight 0.15)  
trustScore: 0-1 (weight 0.2)  
scannerPassRate: 0-1 (weight 0.15)  
Optional: mlScore (weight varies)  
theta: weighted composite 0-1

---

## 20. /aepassist

Interactive assistant. One slash command.

8 modes:  
setup: first-time project setup (3 questions)  
status: current governance state



preset: switch preset (strict, standard, relaxed, audit)  
emergency: kill switch, kill-rollback, pause, resume  
covenant: list, create, view  
identity: show, create (Ed25519), export  
report: JSON, CSV, HTML audit reports  
help: main menu

Available via:  
CLI: npx aep assist [command]  
MCP: /aepassist [command] in Claude Code  
Direct: node dist/cli.js assist [command]

---

## 21. GOVERNANCE PRESETS

strict:  
Trust 200. Human gates. Post-quantum signatures. All 11 scanners (hard).  
Recovery max 1 attempt. Workflow phases required. Fleet max 3 agents.

standard:  
Trust 500. Webhook gates. Drift warnings. 7 core scanners.  
Recovery max 2 attempts. Workflow phases optional.

relaxed:  
Trust 600. No gates. Basic ledger. 4 core scanners (PII, injection, secrets, jailbreak). No recovery. No workflow phases.

audit:  
Read-only. No mutations. Full evidence collection. All 11 scanners (soft).  
Complete OTEL export. Proof bundles for every session.

---

## 22. BUILT-IN POLICIES

Policy	Description
strict-production	Maximum governance for production deployments
coding-agent	Balanced governance for AI coding agents
multi-agent	Cross-agent verification and fleet policies
readonly-auditor	Read-only evidence collection
covenant-only	Covenant enforcement without full policy chain
aep-builder	For agents building AEP-governed systems
full-governance	Everything enabled (scanners, recovery, knowledge, OTEL, tracking)
content-safety	All scanners hard, recovery on, minimal structural governance

---

## 23. SUBPROTOCOLS

6 subprotocols, same architecture applied to different domains:

UI: element types, z-bands, skin bindings, scene graph

Workflows: state transitions, payload schemas, approval gates

REST APIs: HTTP methods, endpoints, bodies, headers

Events: topics, payloads, permissions, correlation IDs

Infrastructure: resource kinds, required fields, type constraints

Commerce: cart, checkout, payment, fulfillment, spend tracking

---

## 24. MULTI-AGENT

Agent identity:

Ed25519 signed. Capability declarations. Covenant references.

Endpoint URLs. Trust tier. Ring assignment.

DNS TXT discovery (\_aep-agent.example.com)

Well-known URL (/well-known/aep-agent.json)

Verification handshake:

Exchange proof bundles before interaction.

Verify: identity signature, expiry, covenant validity, Merkle root.

Any failure blocks with specific reason.

Concurrency control:

Version counter per element. Updates must include expected version.

Conflict on mismatch. Agent must re-read and retry.

---

## 25. INTENT DRIFT

5 heuristics:

Tool category drift

Target scope drift

AEP prefix drift

Frequency anomaly

Repetition detection

Warmup period: default 10 actions (baseline establishment).

Actions on drift: warn, gate, deny, terminate (configurable).

---

## 26. STREAMING VALIDATION

Agent output validated chunk by chunk as it generates.

Checks on accumulated output:

Covenant forbid patterns

Protected elements  
Z-band violations  
Structural errors  
Policy forbidden patterns  
Scanner findings (all 11 scanners)

Hard violation mid-stream: abort immediately.  
Saves tokens and cost by stopping bad output early.

---

## **27. KILL SWITCH**

Emergency termination.  
Options:  
kill: stop all sessions, preserve state  
kill-rollback: stop all sessions, reverse all changes  
All affected agents set to trust 0.  
Every kill logged in evidence ledger.  
Available via /aepassist emergency or npx aep assist kill.

---

## **28. ROLLBACK**

Pre-mutation state stored before every mutation.  
Single action rollback: reverse one change.  
Full session rollback: reverse all changes in reverse chronological order.  
Every rollback logged as compensation entry.

---

## **29. TASK DECOMPOSITION**

Subtask trees with scope inheritance.  
Child scope = intersection of parent scope and declared scope.  
A child can never widen access beyond its parent.  
Completion gates: configurable criteria before task closure.

---

## **30. DATA PROFILING**

Statistical checks on tabular data:  
Null/empty rate (default threshold 30%)  
Duplicate detection (default threshold 10%)  
Outlier detection (z-score, default 4 standard deviations)  
Schema consistency (inconsistent field counts/types)  
Class imbalance (default threshold 90%)

Parses CSV, TSV and JSON array inputs.

CLI: npx aep profile <file>

---

## 31. CLI COMMANDS

npx aep assist [command] /aepassist interactive assistant  
npx aep serve start MCP server (stdio mode)  
npx aep eval <dataset> run evaluation dataset  
npx aep eval --suggest-rules generate rules from eval report  
npx aep dataset create <n> create evaluation dataset  
npx aep dataset add add entry to dataset  
npx aep dataset import-ledger import from production ledger  
npx aep dataset export export to JSON or CSV  
npx aep dataset list list all datasets  
npx aep prompt inject inject governance context  
npx aep prompt compare compare two prompts  
npx aep prompt versions list prompt versions  
npx aep prompt save save prompt version  
npx aep kb create <n> create knowledge base  
npx aep kb ingest <n> <file> ingest file into knowledge base  
npx aep kb query <n> <query> query knowledge base  
npx aep kb stats <n> knowledge base statistics  
npx aep kb list list knowledge bases  
npx aep scan <text-or-file> standalone scanner check  
npx aep call <prompt> governed model call  
npx aep profile <file> data profiling  
npx aep reliability <session> show reliability index  
npx aep commerce status commerce status  
npx aep commerce merchants list merchants  
npx aep commerce spend daily spend tracker  
npx aep fleet status fleet status  
npx aep fleet agents list fleet agents  
npx aep fleet pause pause entire fleet  
npx aep fleet resume resume fleet  
npx aep fleet kill fleet kill switch  
npx aep workflow init <template> create workflow from template

---

## 32. INSTALLATION

Method 1 - Clone: git clone <https://github.com/thePM001/AEP-agent-element-protocol.git> cd  
AEP-agent-element-protocol npm install && npm run build npx aep assist setup

Method 3 - Claude Code MCP:

claude mcp add aep -- node /path/to/dist/cli.js serve

Then: /aepassist setup

---

### **33. RESEARCH**

Paper: "Deterministic Adjudication Lattices and Evolutionary Convergence  
in Constrained Generative Agent Populations"

Author: the.PM / New Lisbon Agency / IPHCCP

Date: April 2026

OpenTimestamps verified

Submitted to cs.AI (cross-listed cs.FL, cs.SE)

Repository: [github.com/thePM001/AEP-research-paper-001](https://github.com/thePM001/AEP-research-paper-001)

---

TRANSHUMAN EUDAIMONIST AI TECHNOLOGY.

BIOSECURITY STATUS AI-ELIGIBILITY CRITERIA APPLY.