Cognous Agent Replay Bundle: Portable Replay Records for AI-Agent Runs

André de Lima
Cognous
Seattle, Washington
andredelima@cogno.us

July 2026

Abstract

AI agents increasingly operate through tools, APIs, data sources, workflow systems, messaging systems, and enterprise applications. As these systems move from demonstration to governed deployment, organizations need more than logs or screenshots. They need portable records that allow an agent run to be reconstructed, validated, redacted, signed, reviewed, and attached to downstream governance evidence. Cognous Agent Replay Bundle is a minimal public reference format for packaging AI-agent runtime records into portable replay artifacts. It provides a JSON-backed schema, Pydantic object model, semantic validator, redaction helper, export-integrity signing helper, command-line utilities, examples, documentation, and tests for representing agent run frames, action proposals, policy decisions, policy evaluation traces, blocked actions, authority records, reliance records, validation reports, redaction metadata, and signature metadata. It is intentionally scoped: it is not an agent framework, not a model runtime, not a replay viewer, not a hosted service, not a security boundary by itself, and not a complete enterprise governance platform. Its contribution is narrower and operational: a structured replay package that helps teams preserve the technical evidence needed to reconstruct what an AI agent proposed, what policy decided, what was blocked, what authority existed, what sources were relied upon, and how the resulting record can be inspected later.

1 Introduction

AI-agent systems are moving from conversational support toward delegated action. Agents can retrieve data, call tools, draft communications, prepare exports, generate recommendations, propose updates, and trigger workflows. As these systems touch more business processes, organizations need a way to reconstruct agent behavior after the fact.

Traditional application logs are useful, but they are rarely sufficient for agent governance. A log line may show that a function was called. It may not show what the agent proposed before execution, what policy decided, what was blocked, what authority existed, what sources the agent relied on, what policy rules were evaluated, whether the run can be replayed, whether sensitive information was redacted, or whether the exported record can be checked for modification.

Agent Replay Bundle addresses this reconstruction problem. It defines a portable evidence package for AI-agent runs. A replay bundle is not the runtime itself. It is the record package produced by or around a runtime system. Its purpose is to make a run inspectable after execution.

The guiding design principle is:

> No enterprise agent run should be considered reviewable unless the organization can reconstruct what the agent proposed, what policy decided, what was blocked, what authority existed, what sources were relied upon, and what output was produced.

The public reference implementation exposes a small set of public objects and utilities: `AgentReplayBundle`, `RunFrame`, `ActionProposal`, `PolicyDecision`, `PolicyEvaluationTrace`, `PolicyRuleEvaluation`, `BlockedAction`, `AuthorityRecord`, `RelianceRecord`, `ValidationIssue`, `ValidationReport`, `RedactionMetadata`, `SignatureMetadata`, `SignedReplayBundle`, a command-line interface, JSON schemas, example bundles, validation helpers, redaction helpers, signing helpers, and a test suite.

2 Problem Statement

Enterprise AI-agent governance has a replay gap. Runtime behavior may occur quickly, across multiple tools and systems, under changing policy conditions. After the fact, reviewers need to reconstruct the run. They need a portable package that can answer a precise question:

> What happened during this agent run, and what evidence exists to review it?

This gap appears in several operational forms.

2.1 Logs Are Not Replay Packages

Logs may record low-level events, errors, or service calls. They do not necessarily preserve agent-level action proposals, policy decisions, blocked actions, authority records, reliance records, or policy evaluation traces in one coherent package.

A replay bundle is organized around the run. It collects the evidence needed to reconstruct and inspect the run, rather than leaving reviewers to infer behavior from scattered logs.

2.2 Action Proposals Are Often Lost

For governed agents, the proposal matters. An agent may propose an action that is never executed because policy blocks it or escalates it. If only executed actions are logged, reviewers lose evidence of what the agent attempted to do.

Replay bundles preserve action proposals regardless of whether the action was allowed, blocked, or escalated.

2.3 Blocked Actions Need First-Class Records

Blocked actions are not failures to be hidden. They are governance evidence. A blocked outbound email, blocked purchase order, blocked data export, or blocked deletion may demonstrate that controls operated as intended.

Agent Replay Bundle records blocked actions explicitly and links them to the corresponding action proposals and policy decisions.

2.4 Policy Decisions Need Context

A policy decision is not just an outcome. Reviewers need to know which action it applied to, what result was returned, which policy produced it, why the decision was made, when it occurred, and whether a corresponding trace exists.

Replay bundles record policy decisions and optionally link them to policy evaluation traces.

2.5 Policy Evaluation Needs Traceability

A decision such as `block` or `allow` is easier to review when the rules leading to that result are preserved. A policy evaluation trace records individual rule evaluations, matched conditions, rule results, explanations, final result, and deterministic fingerprint.

This allows reviewers to inspect not only the final decision, but the decision path.

2.6 Authority and Reliance Are Often Reconstructed Too Late

After an incident or audit, teams may ask whether the agent had authority and what sources it relied on. If those records were not captured during or near the run, reconstruction becomes costly and unreliable.

Agent Replay Bundle includes authority records and reliance records as part of the run package.

2.7 Exports Need Redaction and Integrity Metadata

Run records may contain sensitive payloads, targets, reasons, or final outputs. They may need to be shared with internal reviewers, auditors, customers, counsel, or regulators. Those exports require redaction metadata and integrity metadata.

Agent Replay Bundle includes redaction metadata and export-integrity signing metadata to support safer review workflows.

3 Design Goals

Agent Replay Bundle is designed around ten goals.

1. **Run-centered packaging.** Records should be organized around a specific agent run.
2. **Proposal preservation.** Action proposals should be preserved whether allowed, blocked, or escalated.
3. **Policy decision capture.** Policy decisions should be explicit, inspectable, and linked to action proposals.
4. **Traceability.** Policy evaluation traces should preserve rule-level evaluation detail where available.
5. **Blocked-action visibility.** Blocked actions should be first-class records.
6. **Authority visibility.** Authority records should be associated with the run.
7. **Reliance visibility.** Source reliance should be captured and linked to actions where possible.
8. **Semantic validation.** Bundles should be checked for internal consistency beyond schema validation.
9. **Safe export preparation.** Bundles should support redaction while preserving governance-relevant structure.
10. **Export integrity.** Bundles should support signing for export integrity, while avoiding claims of production key management.

4 Non-Goals

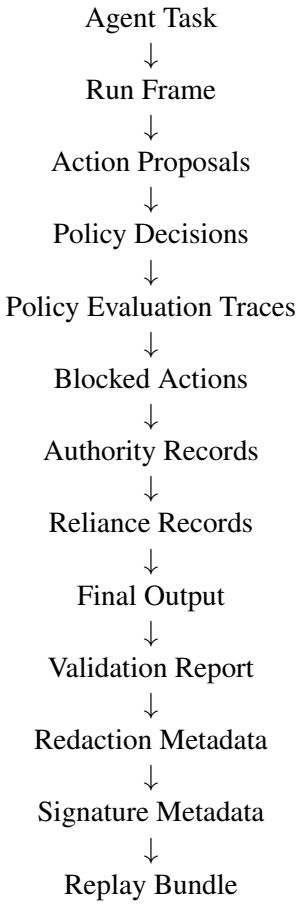
The architecture is deliberately bounded. Agent Replay Bundle is not:

- an agent framework;
- a model runtime;
- a planner;
- a policy engine;
- a runtime control plane;
- a replay viewer;
- a hosted service;
- a complete enterprise governance platform;
- a production authorization system;
- a security boundary by itself;
- a substitute for application-level authorization;
- a guarantee that model outputs are true, safe, or complete;
- a guarantee that regulatory obligations have been satisfied.

It packages run evidence and validates internal consistency. It does not execute actions, enforce policy, govern tool access, certify compliance, or prove that a deployment is safe.

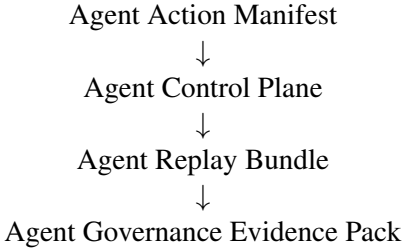
5 System Overview

The replay packaging pattern is:



The bundle may be produced directly by an agent control system, assembled from logs and records, or constructed by another runtime governance layer. It can then be validated, summarized, redacted, signed, stored, shared, or referenced by a business-facing evidence package.

In the Cognous Open Control Stack, Agent Replay Bundle is the *Replay* layer:



The manifest declares what an agent may propose. The control plane records and governs what the agent actually proposes at runtime. The replay bundle packages technical run records. The evidence pack translates those records into business-facing governance evidence.

6 Core Object Model

6.1 AgentReplayBundle

An `AgentReplayBundle` is the top-level object. It packages the run frame, action proposals, policy decisions, traces, blocked actions, authority records, reliance records, final output, validation report, redaction metadata, signature metadata, and optional metadata.

Field	Meaning
`bundle_id`	unique replay bundle identifier
`bundle_version`	replay bundle format version
`run_id`	unique identifier of the agent run
`status`	complete, partial, or redacted
`generated_at`	timestamp when the bundle was generated
`producer`	optional system that produced the bundle
`frame`	task frame describing the run context
`action_proposals`	proposed actions made by the agent
`policy_decisions`	policy decisions made in response to proposals
`policy_traces`	detailed policy evaluation traces
`blocked_actions`	actions blocked by policy
`authority_records`	authority grants in scope for the run
`reliance_records`	sources the agent relied on
`final_output`	final output produced by the run, if available
`validation_report`	optional embedded validation report
`redaction_metadata`	redaction status and redacted field paths
`signature_metadata`	optional embedded signature metadata
`metadata`	optional extension metadata

6.2 RunFrame

A `RunFrame` describes the task context for a run. It records the frame ID, task, actor, environment, allowed tools, blocked tools, policy version, and creation timestamp.

The frame anchors the run. Without it, individual proposals and decisions are difficult to interpret.

6.3 ActionProposal

An `ActionProposal` records an action the agent proposed. It includes the action ID, run ID, tool name, action type, target, payload, timestamp, and optional reason.

Action proposals are preserved even if the action is never executed. This is central to governed replay.

6.4 PolicyDecision

A `PolicyDecision` records the result of evaluating an action proposal. It includes the decision ID, action ID, run ID, result, policy name, reason, decision timestamp, deterministic fingerprint, and optional trace ID.

Policy decisions support three outcomes:

- `allow`;
- `block`;

- escalate.

6.5 PolicyEvaluationTrace

A PolicyEvaluationTrace records detailed policy evaluation. It includes the trace ID, run ID, action ID, evaluation timestamp, list of rule evaluations, final result, and deterministic fingerprint.

The trace allows later review of how a decision was reached.

6.6 PolicyRuleEvaluation

A PolicyRuleEvaluation records the evaluation of one rule. It includes the rule name, whether the rule matched, the rule result, and a reason.

Rule results include:

- allow;
- block;
- escalate;
- none.

6.7 BlockedAction

A BlockedAction records an action that was blocked by policy. It includes the blocked ID, action ID, run ID, reason, policy name, and blocked timestamp.

The validator expects blocked actions to correspond to policy decisions with result block for the same action ID.

6.8 AuthorityRecord

An AuthorityRecord records authority available to the actor during the run. It includes the authority ID, run ID, actor, scope, source, optional expiration timestamp, and creation timestamp.

Authority records do not grant authority by themselves. They preserve evidence of asserted authority context for review.

6.9 RelianceRecord

A RelianceRecord records a source the agent relied on. It includes the reliance ID, run ID, source name, source type, scope, optional referenced action ID, and creation timestamp.

Source types include:

- tool;
- database;
- file;
- api;
- user_input;
- model_output;
- other.

6.10 ValidationIssue and ValidationReport

A `ValidationIssue` records an error or warning found during bundle validation. A `ValidationReport` aggregates validation issues and marks the bundle as valid only when no error-severity issues are present.

6.11 RedactionMetadata

`RedactionMetadata` records whether redaction was applied, when it was applied, which fields were redacted, the redaction policy, replacement value, and optional notes.

6.12 SignatureMetadata

`SignatureMetadata` records whether an export integrity signature exists, when it was applied, the signature algorithm, the signature value, and an optional key ID.

6.13 SignedReplayBundle

A `SignedReplayBundle` wraps an `AgentReplayBundle` with authoritative signature metadata. The signed wrapper is the preferred signed-export object. Embedded signature metadata on `AgentReplayBundle` is optional and is not part of the canonical signing payload.

7 Action Types

The replay bundle uses action types to classify agent behavior. The current reference implementation supports:

Action Type	Meaning
`read`	retrieves or inspects information
`write`	creates or modifies information
`external_send`	sends information outside the immediate system boundary
`delete`	removes information or records
`export`	packages or transfers information for downstream use
`purchase`	initiates or prepares a purchase action
`approve`	approves, confirms, or authorizes a workflow step
`escalate`	escalates to a human, queue, or supervisory process
`other`	action type not otherwise classified

These types are deliberately simple. Their purpose is not to define a complete enterprise ontology. Their purpose is to provide enough structure for run review, validation, and governance evidence.

8 Bundle Status

A replay bundle supports three status values:

Status	Meaning
`complete`	run package is intended to represent a completed run
`partial`	run package is incomplete or still in progress
`redacted`	run package has been redacted for export or review

A complete bundle without final output currently produces a warning rather than an error. This allows the format to support operational cases where final output is unavailable, unavailable by policy, or captured elsewhere.

9 Semantic Validation

The current public reference implementation validates bundle consistency and emits a `ValidationReport`. Errors invalidate the bundle. Warnings indicate review concerns but do not make the bundle invalid.

9.1 Error Conditions

The validator checks error conditions including:

1. `bundle_id` must be non-empty.
2. `bundle_version` must be non-empty.
3. `run_id` must be non-empty.
4. `frame.frame_id` must be non-empty.
5. `frame.actor` must be non-empty.
6. `frame.policy_version` must be non-empty.
7. every `ActionProposal.run_id` must equal `bundle.run_id`.
8. every `PolicyDecision.run_id` must equal `bundle.run_id`.
9. every `PolicyDecision.action_id` must reference an existing `ActionProposal.action_id`.
10. every `BlockedAction.run_id` must equal `bundle.run_id`.
11. every `BlockedAction.action_id` must reference an existing `ActionProposal.action_id`.
12. every `BlockedAction` must correspond to at least one `PolicyDecision` with result `block` for the same `action_id`.
13. every `AuthorityRecord.run_id` must equal `bundle.run_id`.
14. every `RelianceRecord.run_id` must equal `bundle.run_id`.
15. every `RelianceRecord.referenced_action_id`, when present, must reference an existing `ActionProposal.action_id`.
16. every `PolicyEvaluationTrace.run_id` must equal `bundle.run_id`.
17. every `PolicyEvaluationTrace.action_id` must reference an existing `ActionProposal.action_id`.
18. every `PolicyDecision.trace_id`, when present, must reference an existing `PolicyEvaluationTrace.trace_id`.

19. when `trace_id` is present, the trace deterministic fingerprint must match the decision deterministic fingerprint.
20. when `trace_id` is present, the trace final result must match the decision result.
21. `E021` is reserved for complete-bundle final-output checks.
22. if `signature_metadata.signed` is true, `signature_algorithm` must be present.
23. if `signature_metadata.signed` is true, `signature` must be present.
24. if `redaction_metadata.redacted` is true, `redacted_at` must be present.
25. if `redaction_metadata.redacted` is true, `redacted_fields` must not be empty.

9.2 Warning Conditions

The validator checks warning conditions including:

- bundle has no action proposals;
- bundle has no policy decisions;
- bundle has no policy evaluation traces;
- bundle has no reliance records;
- bundle has no authority records;
- `frame.allowed_tools` is empty;
- `frame.blocked_tools` is empty;
- an action proposal has no reason;
- a policy decision has no `trace_id`;
- bundle status is `complete` but `final_output` is missing;
- an action payload contains potentially sensitive keys but `redaction_metadata` is not present;
- `signature_metadata` is missing;
- embedded `validation_report.checked_id` does not match `bundle_id`.

Validation remains implementation-level. It does not prove that the bundle is complete, that the underlying records are true, that policy was sufficient, or that compliance obligations have been satisfied.

10 Redaction

Replay bundles may contain sensitive fields. Action payloads, action targets, final outputs, and reasons may include personal data, customer identifiers, operational details, or confidential information. The public reference implementation includes a redaction helper for preparing safer exports.

The redaction helper:

- returns a deep copy and does not mutate the original bundle;
- can redact action payloads;
- can redact action targets;
- can redact final output;
- can redact reason fields on action proposals, policy decisions, policy rule evaluations, and blocked actions;
- preserves IDs, timestamps, decision results, policy names, deterministic fingerprints, and cross-object references;
- changes bundle status to `redacted`;
- records redacted field paths in `redaction_metadata`;
- records redaction timestamp, policy name, replacement value, and notes where available.

Redaction is not a guarantee that every sensitive field has been removed. It must be configured correctly and reviewed under the relevant data-handling policy. The helper is a reference implementation, not a complete data-loss-prevention system.

11 Signing

The public reference implementation includes HMAC-SHA256 export-integrity signing. Signing is used to help detect modification of exported bundles. It is not production key management and does not provide a complete trust framework.

11.1 Canonical Serialization

The signing helper creates a canonical JSON representation of an `AgentReplayBundle` by:

1. calling `bundle.model_dump(mode="json");`
2. removing the entire `signature_metadata` field from the bundle payload;
3. serializing with sorted keys, compact separators, and ASCII-safe encoding.

Removing embedded signature metadata ensures that embedded signature state does not affect signing or verification. The same bundle produces the same canonical string when all non-signature fields match.

11.2 SignedReplayBundle

The signing helper returns a `SignedReplayBundle`. In the signed wrapper, `SignedReplayBundle.signature_metadata` is the authoritative signature metadata for signed exports.

`AgentReplayBundle.signature_metadata` remains optional embedded metadata for systems that store signature state directly on the bundle. It is not part of the canonical signing payload.

11.3 Limitations

HMAC-SHA256 with a shared secret provides export integrity under the assumption that the secret is handled securely. It does not provide:

- non-repudiation;
- a public-key trust hierarchy;
- protection against compromise of the signing secret;
- production key rotation;
- certificate authority semantics;
- regulatory acceptance by itself.

For high-assurance production use cases, implementers should use appropriate key management systems and may consider asymmetric signing with an HSM or KMS.

12 Command-Line Interface

The `arb` command-line interface provides utilities for local validation, summarization, redaction, signing, verification, and example checking.

12.1 Implemented Commands

The current public reference implementation includes:

- `arb validate path/to/replay_bundle.json`;
- `arb summarize path/to/replay_bundle.json`;
- `arb redact path/to/replay_bundle.json --out path/to/redacted.json [--targets] [--final-output] [--reasons] [--policy "policy-name"]`;
- `arb sign path/to/replay_bundle.json --secret SECRET --out path/to/signed.json [--key-id KEY_ID]`;
- `arb verify path/to/signed_replay_bundle.json --secret SECRET`;
- `arb check-examples`.

The validation command loads a bundle, runs semantic validation, prints errors and warnings, and returns a process exit code. The summarize command prints a compact run summary. The redact command produces a redacted bundle. The sign command creates a signed wrapper. The verify command checks a signed wrapper. The check-examples command validates included example bundles.

These commands are intended for local validation, examples, and lightweight workflows. They are not a hosted service, daemon, dashboard, or enterprise control surface.

13 JSON Schemas and Public Conformance

The repository includes JSON Schema Draft 2020-12 files for the public replay-bundle objects. The primary schema is `agent_replay_bundle.schema.json`. These schemas define the public interchange boundary for records such as `AgentReplayBundle`, `RunFrame`, `ActionProposal`, `PolicyDecision`, `PolicyEvaluationTrace`, `PolicyRuleEvaluation`, `BlockedAction`, `AuthorityRecord`, `RelianceRecord`, `ValidationReport`, `RedactionMetadata`, `SignatureMetadata`, and `SignedReplayBundle`.

The schemas serve four purposes:

1. **Interchange.** They allow external systems to understand the shape of replay-bundle records.
2. **Validation.** They support structural validation outside the Python package.
3. **Documentation.** They make the public replay boundary explicit.
4. **Conformance.** They provide a basis for future compatibility tests across implementations.

The schema boundary is deliberately limited. JSON schemas validate structure, required fields, field types, and allowed values where specified. They do not prove that a replay bundle is operationally complete, that runtime controls were sufficient, or that a compliance obligation has been satisfied.

14 Example Bundles

The public repository includes example replay bundles for common enterprise agent scenarios.

14.1 Customer-Service Replay Bundle

A customer-service agent reads CRM data and proposes an outbound customer email. The replay bundle records the run frame, action proposal, policy decision, blocked action, authority context, reliance record, and final output indicating that the email was blocked due to missing external-send authority.

14.2 Internal Research Replay Bundle

An internal research agent searches documents, reads files, and proposes export behavior. The replay bundle records document-retrieval reliance, policy decisions, and escalation behavior for export review.

14.3 Procurement Replay Bundle

A procurement agent compares vendors, reviews quote information, and proposes purchase-order behavior. The replay bundle records proposed actions, policy decisions, blocked purchase behavior, authority context, and reliance sources.

14.4 Redacted Replay Bundle

A redacted example shows how sensitive payloads and final output can be removed while preserving governance-relevant structure, identifiers, decision results, references, and metadata.

14.5 Signed Replay Bundle

A signed example shows the structure of a `SignedReplayBundle` with export-integrity metadata. The example is for demonstration and structural validation, not production key management.

15 Relationship to Runtime Control

Agent Replay Bundle is not a runtime control layer. It depends on evidence produced by runtime systems, agent frameworks, logs, control planes, or other governance infrastructure.

A runtime control layer can populate a replay bundle by recording:

- the task frame;
- proposed actions;
- policy decisions;
- policy evaluation traces;
- blocked actions;
- authority records;
- reliance records;
- final output;
- validation reports;
- redaction metadata;
- signing metadata.

In the Cognous public reference stack, Agent Control Plane is one source of records that can be packaged into an Agent Replay Bundle. Other systems can also produce replay bundles if they implement the public format.

16 Relationship to ODES

ODES, the Open Decision Evidence Standard, is a separate proposed open, vendor-neutral standard for portable decision evidence in AI-mediated and cross-boundary decisions. Agent Replay Bundle is not the ODES standard and does not define ODES conformance.

The relationship is optional and implementation-adjacent. A replay bundle may contain technical run evidence that could support or be referenced by an ODES-compatible portable decision evidence record in scenarios where an agent-mediated decision crosses an organizational, system, or jurisdictional boundary.

The recommended boundary is:

- Agent Replay Bundle remains a native replay artifact for AI-agent runs.
- ODES remains a vendor-neutral proposed standard for portable decision evidence records.
- Future tools may provide informative mappings or optional export paths from replay-bundle data to ODES-style records.
- No conformance to ODES should be claimed unless an ODES profile and conformance test suite define that claim.

This preserves the distinction between operational replay artifacts and public standards work.

17 Security Considerations

Agent Replay Bundle improves reconstructability and evidence handling, but it should not be treated as a security product.

Important limitations include:

- the bundle records evidence but does not enforce policy;
- authority records preserve asserted authority context but do not grant authority;
- reliance records document sources but do not prove those sources were correct;
- policy traces record rule evaluation but do not prove policy sufficiency;
- redaction helpers must be configured and reviewed correctly;
- signing provides export integrity, not production key management;
- validation checks structure and consistency, not truth or completeness;
- production deployments still require authentication, authorization, monitoring, incident response, runtime control, and organizational governance.

The correct security posture is layered. Agent Replay Bundle supplies structured replay evidence. It should be combined with application authorization, runtime policy gates, secure logging, monitoring, redaction workflows, key management, operational controls, and governance review.

18 Implementation Boundary

The public reference implementation is intentionally thin. It provides enough structure to demonstrate the replay-bundle category and support real integration experiments. It does not reveal or depend on private Cognous product internals.

The public boundary is therefore:

Public Reference Layer	Private Product Layer
Replay bundle schema	hosted replay registry
Pydantic object model	enterprise replay store
Semantic validator	automated governance workflows
Redaction helper	production redaction workflows
HMAC signing helper	production key management
CLI utilities	platform APIs and managed services
Example bundles	deployment-specific templates
Policy decision records	enterprise policy authoring systems
Policy evaluation traces	runtime policy analytics
Blocked-action records	dashboarding and alerting
Authority records	customer-specific authority systems
Reliance records	production provenance pipelines
Validation reports	compliance evidence automation
Signed wrappers	managed export workflows
JSON schemas	enterprise conformance tooling

Public software should expose the integration contract: what run evidence can be packaged, how it can be validated, how sensitive fields can be redacted, and how export integrity can be represented. Private product systems can add enterprise integrations, hosted workflows, identity management, richer policy authoring, dashboards, evidence registries, workflow routing, and customer-specific governance logic.

19 Reference Implementation Status

The public reference implementation is intentionally small. It is meant to be inspectable, testable, and safe to publish.

Capability	Status	Notes
Core object model	Implemented	Pydantic v2 models
Replay bundle loader	Implemented	JSON file loading and model validation
Replay bundle dumper	Implemented	formatted JSON export
Run frame	Implemented	task, actor, tools, policy version
Action proposals	Implemented	tool, type, target, payload, reason
Policy decisions	Implemented	allow, block, escalate outcomes
Policy evaluation traces	Implemented	rule-level evaluation detail
Blocked actions	Implemented	explicit blocked-action records
Authority records	Implemented	authority context for run
Reliance records	Implemented	source reliance evidence
Validation reports	Implemented	error and warning issues
Semantic validation	Implemented	cross-reference and consistency checks
Redaction helper	Implemented	payload, target, reason, output redaction
Redaction metadata	Implemented	redacted field paths and policy details
Signing helper	Implemented	HMAC-SHA256 export integrity
Canonical signing payload	Implemented	excludes embedded signature metadata
Signed replay bundle	Implemented	wrapper-level signature metadata
CLI validation	Implemented	`arb validate`
CLI summary	Implemented	`arb summarize`
CLI redaction	Implemented	`arb redact`
CLI signing	Implemented	`arb sign`
CLI verification	Implemented	`arb verify`
Example validation	Implemented	`arb check-examples`
JSON schemas	Implemented	public schema boundary
Example bundles	Implemented	customer service, research, procurement, redacted, signed
Test suite	Implemented	model, loader, validator, redaction, signing, CLI, examples
Hosted replay viewer	Extension	out of current minimal scope
Production key management	Extension	out of current minimal scope
Enterprise replay registry	Extension	out of current minimal scope
ODES export mapping	Extension	optional future compatibility layer

20 Test Suite and Continuous Integration

The repository includes a test suite covering the public reference behavior. Tests are not part of the replay format itself, but they are part of the public trust surface. They show which claims are exercised by executable checks.

The test suite covers areas such as:

- valid replay-bundle loading;
- invalid JSON behavior;
- model validation behavior;
- required field validation;
- run ID consistency;
- action-decision cross-references;
- blocked-action consistency;
- authority-record consistency;
- reliance-record consistency;
- policy-trace consistency;
- fingerprint and final-result matching;
- warning/error distinction;
- redaction deep-copy behavior;
- redaction metadata behavior;
- signing and verification;
- canonical signing stability;
- exclusion of embedded signature metadata from canonical signing payload;
- CLI validation;
- CLI summarization;
- CLI redaction;
- CLI signing;
- CLI verification;
- example bundle validation.

A GitHub Actions workflow runs the test suite and validates example bundles across supported Python versions. This gives the public reference implementation a basic compatibility and regression signal without implying production certification.

21 Example Use Cases

21.1 Agent Run Review

An AI platform team reviews a completed customer-service agent run. The replay bundle shows the task, allowed tools, blocked tools, proposed CRM read, proposed email send, policy decision, blocked outbound send, reliance on CRM data, and final output. Reviewers can reconstruct the run without searching across raw logs.

21.2 Incident Investigation

A customer-facing incident raises questions about whether an agent attempted to send an unauthorized message. The replay bundle preserves both the proposal and the block decision. Investigators can determine what was proposed, which policy blocked it, and what final output was returned.

21.3 Audit Preparation

An audit team needs evidence that runtime controls operated. Replay bundles provide run-level artifacts showing action proposals, policy decisions, traces, blocked actions, authority records, reliance records, validation results, redaction metadata, and signatures.

21.4 Evidence Pack Generation

A governance team prepares a business-facing evidence pack. Replay bundles provide the technical records that support summaries of blocked actions, reliance sources, validation status, redacted exports, and signed artifacts.

21.5 Controlled Export

A bundle must be shared with reviewers outside the engineering team. Sensitive payloads and final output are redacted while IDs, timestamps, decision results, fingerprints, and references are preserved. The redacted export is signed for integrity.

22 Evaluation Criteria

The usefulness of Agent Replay Bundle should be evaluated against operational criteria rather than benchmark scores.

A deployment should ask:

1. Does the bundle identify the run being reviewed?
2. Does it preserve the task frame?
3. Does it record action proposals before execution?
4. Does it record policy decisions for those proposals?
5. Does it link decisions to proposals?
6. Does it preserve blocked actions as first-class records?

7. Does each blocked action correspond to a block decision?
8. Does it record authority context?
9. Does it record reliance sources?
10. Does it include policy evaluation traces where available?
11. Do traces match decision fingerprints and final results?
12. Does it preserve final output when appropriate?
13. Can it be semantically validated?
14. Can it be redacted without breaking governance structure?
15. Can it be signed for export integrity?
16. Can it be verified after export?
17. Can it support business-facing evidence generation?
18. Can it support incident review or audit preparation?

23 Roadmap

The public roadmap should remain implementation-focused and avoid overclaiming.

Implemented MVP capabilities include:

- replay bundle object model;
- run frames;
- action proposals;
- policy decisions;
- policy evaluation traces;
- blocked actions;
- authority records;
- reliance records;
- validation reports;
- semantic validation;
- redaction helper;
- redaction metadata;
- HMAC-SHA256 signing helper;
- canonical signing payload;

- signed replay bundle wrapper;
- command-line validation;
- command-line summaries;
- command-line redaction;
- command-line signing;
- command-line verification;
- example bundles;
- JSON schemas;
- test suite and continuous integration.

Near-term extensions may include:

- richer evidence references;
- additional redaction profiles;
- stronger signing profiles;
- optional asymmetric signature examples;
- replay bundle diffing;
- replay summary rendering;
- optional import helpers from control-plane records;
- optional export helpers to business-facing evidence packs;
- informative ODES mapping documentation;
- additional industry examples;
- conformance test suite.

Longer-term work may include hosted replay viewers, enterprise replay registries, production key management integrations, search and investigation interfaces, workflow integrations, dashboarding, compliance mappings, and managed export workflows. Those are product-layer concerns and should remain separate from the minimal public reference implementation.

24 Conclusion

AI-agent governance requires reconstructability. As agents propose actions, use tools, rely on sources, and interact with enterprise systems, organizations need portable evidence packages that preserve the run context, proposed actions, policy decisions, traces, blocked actions, authority context, reliance records, final output, validation status, redaction metadata, and export integrity metadata.

Cognous Agent Replay Bundle provides a minimal public reference architecture for that replay layer. It describes run frames, action proposals, policy decisions, policy evaluation traces, blocked actions, authority records, reliance records, validation reports, redaction metadata, signature metadata, and signed replay bundles. It validates internal consistency, publishes JSON schemas for the public replay boundary, includes example bundles for common enterprise scenarios, and provides command-line utilities for validation, summarization, redaction, signing, verification, and example checking.

The contribution is deliberately narrow. Agent Replay Bundle does not run agents, enforce policies, certify compliance, or prove correctness. It supplies a concrete technical replay artifact for a specific operational need: making AI-agent runs reconstructable, portable, redaction-ready, signature-ready, and usable as supporting evidence for governed deployment.

References

[1] Cognous. *Agent Replay Bundle: Portable replay records for AI-agent runs*. Public reference implementation, 2026. `https://github.com/cogno-us/cognous-agent-replay-bundle`

[2] Cognous. *Agent Action Manifest: Declare what actions an AI agent may propose before runtime execution*. Public reference implementation, 2026. `https://github.com/cogno-us/cognous-agent-action-manifest`

[3] Cognous. *Agent Control Plane: Runtime control and replay for AI agents*. Public reference implementation, 2026. `https://github.com/cogno-us/cognous-agent-control-plane`

[4] Cognous. *Agent Governance Evidence Pack: Business-facing evidence packages for governed AI-agent deployment*. Public reference implementation, 2026. `https://github.com/cogno-us/cognous-agent-governance-evidence-pack`

[5] Open Decision Evidence Standard. *A Portable, Vendor-Neutral Record Format for Decision Evidence in AI-Mediated and Cross-Boundary Decisions*. Discussion draft, July 2026. `https://opendecisionevidence.org`

[6] National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework*. NIST AI 100-1, 2023.

[7] OWASP Foundation. *OWASP Top 10 for Large Language Model Applications*. Project documentation.

[8] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.