BUSINESS EXPLAINER

Cognous Agent Action Manifest

Declare what AI agents may propose before runtime execution

> *Before an enterprise can control what an AI agent does, it must declare what that agent may propose.*

Business explainer prepared from the Cognous Agent Action Manifest whitepaper (de Lima, 2026).

1. Executive Summary

Cognous Agent Action Manifest is an open-source reference format for declaring what actions an AI agent may propose — before the agent ever runs. It helps teams create a structured action inventory that describes the agent's tools, its specific actions, the type of each action, which actions require authority, which require human review, which should produce reliance evidence, what payload fields matter, and which fields should be redacted in downstream records.

It is deliberately limited in what it claims to be. It does not run the agent. It does not enforce runtime policy. It does not replace authorization. What it does is create the pre-runtime declaration layer that governed deployment depends on: a reviewable, portable statement of the agent's action surface, produced before the agent touches enterprise systems.

The central message for enterprise leaders is this:

> *As enterprises move from AI-agent demos to production agents that use tools, access systems, and trigger workflows, governance cannot begin only at runtime. Enterprises first need to know what the agent is allowed to propose.*

A chatbot can be reviewed by looking at what it said. An agent must also be understood by looking at what it can do. Most organizations today cannot answer the question "what is this agent allowed to propose?" with anything more structured than a prompt, a tool list, and institutional memory. Agent Action Manifest makes that action surface explicit — in a format that business, security, governance, and engineering teams can all read, review, and validate before deployment.

2. The Business Problem

AI agents are beginning to:

- call tools and APIs;
- access customer, financial, operational, or internal data;
- draft and send communications;
- prepare exports;
- propose database updates;
- approve or escalate workflow steps;
- generate recommendations that influence decisions.

This creates a pre-runtime governance gap. An organization may know an agent's general purpose — "it handles customer inquiries," "it summarizes research," "it drafts purchase orders" — without having any structured record of the specific actions it may propose, the tools those actions use, which actions require authority, which require review, which should produce provenance evidence, and which payload fields may expose sensitive data.

In practice, this information usually exists — but it is scattered across prompts, code, tool wrappers, configuration files, permission systems, documentation, and informal team knowledge. Nobody can review it as a whole, because it does not exist as a whole. That gap shows up as six recurring pain points.

1. Unclear agent action surface. Teams know the agent's purpose but not the full set of actions it may propose. Purpose statements and prompts are not inventories.

2. Tool access without action semantics. A tool list shows what systems the agent can call, but not what it will do with them. Access to a CRM does not distinguish reading a record from updating one, and access to an email system does not distinguish drafting a message from sending it. Reads, writes, exports, sends, approvals, deletions, and purchases do not carry the same risk — yet a tool list treats them identically.

3. Ambiguous authority requirements. Privileged actions may not be clearly tied to the authority scopes they should require. When authority expectations are undeclared, runtime systems and application teams end up inventing permission logic on the fly — often inconsistently, often too broadly.

4. Inconsistent human review. One team requires approval before outbound communication; another relies on the agent's own judgment; a third assumes someone else is checking. Review and approval expectations live in prompts, code comments, policy documents, and team assumptions — everywhere except a reviewable artifact.

5. Missing reliance expectations. Some actions should produce evidence of what the agent depended on — which file, database, API, or user input stood behind a summary or a recommendation. If nobody specifies which actions require that evidence, runtime provenance ends up incomplete or inconsistent.

6. Unsafe payload handling. Agent actions carry payloads: customer identifiers, email addresses, account notes, invoice numbers, message bodies. If sensitive fields are not identified before runtime, they surface later — in logs, replay records, and evidence exports that were never prepared to handle them.

These pain points are concrete. Consider five common deployments:

- A **customer-service agent** reads CRM records and sends customer emails.
- An **internal research agent** searches files and exports a summary.
- A **procurement agent** drafts a purchase order.
- A **finance workflow agent** proposes payment approval.
- A **support agent** escalates a customer case.

Each of these agents has an action surface that mixes routine reads with privileged operations. Without a structured declaration, the organization discovers the shape of that surface during deployment — or during an incident — rather than before either.

3. What Agent Action Manifest Is

Agent Action Manifest is a structured declaration of an AI agent's action surface, created before runtime. It describes what the agent may propose, what tools those actions use, what authority is required, what review posture applies, what reliance evidence is expected, and what payload fields may need redaction.

In one sentence:

> *It gives enterprises a structured answer to the question: "What is this agent allowed to propose?"*

The word "manifest" is worth translating into business terms. An agent action manifest is:

- **a pre-runtime action inventory** — every declared action, classified by type;
- **a portable declaration of agent capabilities** — a single artifact instead of scattered fragments;
- **a governance preparation artifact** — the input that runtime controls and evidence workflows can build on;

- **a reviewable record before deployment** — something security, risk, and governance teams can inspect and sign off on;
- **not runtime enforcement by itself** — it declares posture; it does not execute policy.

Where it fits: the Cognous Open Control Stack

Agent Action Manifest is the first layer of a four-layer pattern for governed agent deployment:

Declare → Control → Replay → Evidence

- **Agent Action Manifest** declares what an agent may propose, before runtime.
- **Agent Control Plane** records and governs what the agent actually proposes at runtime.
- **Agent Replay Bundle** packages runtime records so runs can be reconstructed.
- **Agent Governance Evidence Pack** translates runtime evidence into business-facing review material.

The distinction between the first two layers is the one that matters most. The manifest is the Declare layer: it states the expected action surface before anything runs. The control plane is the Control layer: it records and governs actual behavior during runs. Declaration without runtime control is a plan with no enforcement; runtime control without declaration is enforcement with no baseline. The two are designed to work together — and the manifest gives runtime governance something concrete to govern against.

4. What It Does

Declares the agent's tools

The manifest identifies the tools or systems the agent may use — the CRM, the document store, the email system, the workflow engine — including whether each is linked to an external system and how its data is classified.

Business value: teams can see which external systems, APIs, files, or databases are part of the agent's operating environment, in one place.

Declares specific actions

The manifest separates tools from actions. A single tool may support many actions, and not all actions carry the same risk: the same email tool can draft a message or send one.

Business value: the enterprise can distinguish a read from a write, a draft from a send, and an export from an approval — distinctions a tool list cannot make.

Classifies action types

Every declared action is classified: read, write, external send, delete, export, purchase, approve, escalate, or other.

Business value: risk and governance teams can identify privileged or sensitive action categories at a glance, rather than inferring them from code or prompts.

Declares authority requirements

The manifest identifies which actions require authority scopes — for example, that sending customer email requires an outbound-send permission, or that creating a purchase order requires a procurement scope. Declaring a requirement does not grant the authority; it states what should exist before the action is eligible for execution.

Business value: teams define what permission should exist before runtime, instead of discovering missing controls during deployment.

Declares review requirements

The manifest identifies whether an action requires no review, human review, formal approval from a specified reviewer role, or draft-first handling — where the agent prepares output for a person to finalize.

Business value: human oversight expectations become explicit and action-specific, rather than scattered across prompts, code, and team assumptions.

Declares reliance requirements

The manifest indicates whether an action should produce evidence of what it relied on — a tool, database, file, API, user input, or model output.

Business value: provenance expectations are built into the action inventory before runtime, so downstream systems know which actions must leave a source trail.

Declares payload policies

The manifest can specify, for each action's payload, which fields are required, which are optional, which are sensitive, and which are forbidden entirely.

Business value: teams identify sensitive or prohibited data elements before runtime records are created — not after those records already contain them.

Declares redaction hints

The manifest can identify specific fields that should be redacted in downstream records or evidence exports, with a reason and a replacement value.

Business value: governance and audit review can be prepared without overexposing sensitive information. Redaction stops being an afterthought applied to records that were never designed for it.

Validates manifest consistency

A manifest can be validated automatically: missing required fields, duplicate actions, actions that reference undeclared tools, privileged actions with no declared authority, payload-policy conflicts, and approval requirements with no reviewer role all surface as errors or warnings in a validation report.

Business value: teams detect governance gaps before deployment — including the quiet ones, like a high-risk action that defaults to "allow" with no authority and no review posture.

5. Why It Is Valuable

Better deployment readiness. The manifest gives teams a structured view of what the agent can propose before the agent enters production. Deployment review becomes a review of an artifact, not an archaeology of prompts and code.

Better security review. Security teams can see tools, privileged actions, authority requirements, review requirements, and sensitive fields in one place — and can identify which actions need additional runtime controls before signing off.

Better governance consistency. The same action categories and requirement types apply across every agent. Two teams deploying two different agents describe their action surfaces in the same vocabulary, which makes portfolio-level review possible.

Better runtime integration. Runtime control systems receive a clear declaration of expected actions, authority scopes, reliance requirements, and redaction-sensitive fields — instead of being configured from scratch against an undocumented agent.

Better audit preparation. The manifest provides the declared baseline that runtime records, replay bundles, and governance evidence packs can later be compared against. "Did this agent do what it was declared to do?" becomes an answerable question.

Better cross-functional communication. Business, security, governance, and engineering teams review the same artifact instead of each interpreting code, prompts, or documentation differently. The manifest is readable by people who will never open the codebase.

Better separation of responsibilities. Agent developers keep their frameworks and tools. Governance teams get a manifest layer for action inventory and deployment review. Neither has to adopt the other's working environment.

6. How It Is Different from Other Approaches

Enterprises evaluating agent governance already have several artifacts that describe agents in some way. Agent Action Manifest overlaps with none of them exactly, and it is worth being precise about the differences.

Versus agent prompts

Prompts describe what an agent should do, in natural language, primarily to steer model behavior. They are not structured, not validated, and not designed for governance review — and what a prompt requests is not the same as what an agent can propose.

Difference: Prompts guide model behavior. Agent Action Manifest declares the agent's action surface in a structured, reviewable format.

Versus tool lists

Tool lists show what systems an agent can call. They say nothing about what the agent may do through those systems — whether it will read, write, export, send, approve, delete, or purchase.

Difference: Tool lists identify access. Agent Action Manifest identifies action semantics, authority requirements, review posture, reliance expectations, and redaction hints.

Versus API specifications

API specs document what a service can do for any caller. They describe the service's capabilities, not what a particular agent is permitted to propose in a governed enterprise setting.

Difference: API specs document service capabilities. Agent Action Manifest documents the governed action surface of an agent.

Versus policy documents

Policy documents define organizational intent: what kinds of actions require approval, how customer data must be handled. They rarely descend to action-level declarations for a specific agent, and they cannot be machine-validated against an agent's actual configuration.

Difference: Policy documents say what should happen. Agent Action Manifest turns action-level governance posture into a structured, validatable artifact.

Versus runtime control systems

Runtime control systems evaluate and record actual agent behavior during execution — what was proposed, allowed, blocked. They are the enforcement and evidence layer, and they work best when the expected action surface has been declared in advance.

Difference: Runtime control governs what the agent actually proposes. Agent Action Manifest declares what the agent is expected or allowed to propose before runtime.

Versus AI governance platforms

Governance platforms manage programs: model inventories, approvals, risk assessments, dashboards, compliance workflows. That is program-level scope.

Difference: Governance platforms manage the program. Agent Action Manifest provides the pre-runtime action declaration that such platforms can reference — a small artifact, not a competing platform.

Comparison at a glance

Category	What it does	What it misses	How Agent Action Manifest differs
Agent prompts	Guide model behavior in natural language	Structure, validation, action-level governance detail	Declares the action surface in a structured, reviewable, validatable format
Tool lists	Identify which systems an agent can access	Action semantics: read vs. write vs. send vs. approve	Declares specific actions, types, authority, review, reliance, and redaction posture
API specifications	Document what a service can do for any caller	What a specific agent is permitted to propose	Documents the governed action surface of an agent, not the service
Policy documents	Define organizational intent	Action-level, agent-specific, machine-validatable declarations	Turns governance posture into a structured artifact per agent and per action
Runtime control systems	Record and govern actual behavior during runs	A declared baseline of expected actions	Declares the expected action surface that runtime control can govern against
AI governance platforms	Manage programs: inventories, approvals, reporting	Pre-runtime, action-level declarations for individual agents	Supplies the declaration artifact such platforms can reference

Table 1. Agent Action Manifest compared with adjacent artifact categories.

7. What It Is Not

Clear scope is part of the design. Cognous Agent Action Manifest is **not**:

- an agent framework;
- a model runtime;
- a policy engine;

- a runtime control plane;
- a hosted dashboard;
- a security boundary by itself;
- a substitute for application authorization;
- a complete enterprise governance platform;
- a guarantee that model outputs are correct;
- a guarantee that runtime enforcement will occur.

Its value is focused. It creates a structured pre-runtime declaration of an agent's action surface — and a declaration, however well validated, does not enforce itself. Declared authority requirements do not grant or verify authority. Declared review requirements do not guarantee that review occurred. Redaction hints protect nothing unless downstream tooling applies them. The manifest must be combined with runtime controls, application authorization, monitoring, audit logging, and governance review to deliver its full value.

8. Example Scenario

Consider an enterprise preparing to deploy a customer-service agent. The agent can read CRM records, search account notes, draft customer emails, and send approved emails.

1. **The team creates a manifest** for the agent before deployment, naming the agent, its owner, and its deployment environment.
2. **Each tool is declared:** the CRM system, the notes index, and the email system — including which are linked to external systems.
3. **Each action is declared separately.** Reading a CRM record, searching notes, drafting an email, and sending an email are four distinct actions, even though they use only three tools.
4. **The CRM read is classified as a read action** — routine, but expected to leave a provenance trail.
5. **The email draft is classified as draft-first** — the agent prepares content; a person finalizes it.
6. **The email send is classified as an external send** — information leaving the system boundary, the highest-scrutiny category in this deployment.
7. **The email send requires authority:** a declared outbound-send scope must exist before the action is eligible for execution.
8. **The email send requires approval,** with a supervisor named as the reviewer role.
9. **CRM reads and note searches require reliance records,** so that any customer-facing response can later be traced to the sources behind it.

10. **Sensitive fields are marked:** customer ID, email address, and message body are declared sensitive in the payload policies.
11. **Redaction hints identify** which of those fields should be redacted in exported evidence, and what replacement text to use.
12. **The manifest is validated before deployment.** The validation report confirms there are no error-level issues: no privileged action without authority, no approval requirement without a reviewer, no payload conflicts.
13. **The manifest goes to security and governance review** — a single readable artifact, not a code walkthrough.
14. **Later, at runtime,** Agent Control Plane can record what the agent actually proposes and compare runtime behavior against the declared action inventory.

The business outcome: before the agent touches customer systems, the organization has a structured, reviewable declaration of what the agent may propose and what governance posture applies to each action. If the agent later proposes something outside the declared surface, that deviation is detectable — because the surface was declared.

9. Why This Matters Now

Enterprises are moving from experimentation to deployment, and the moment an AI agent gets access to tools, APIs, files, customer records, workflow systems, or communication channels, the risk profile changes. A demo agent's capabilities can live in a prompt. A production agent's capabilities need to live in an artifact the organization can review, validate, and hold the deployment against.

A useful framing for leadership:

- **Chatbots require output review.** The text is the risk surface.
- **Agents require action inventory.** What the agent can propose is the risk surface — and it must be declared to be reviewed.
- **Governed agents require runtime control and replay.** Actual behavior must be recorded and reconstructable.
- **Enterprise agent programs require business-readable evidence.** Boards, auditors, and regulators read artifacts, not codebases.

Runtime governance is weaker when the action surface is undefined. *A runtime control layer can only gate and record against expectations that exist. Agent Action Manifest helps define that surface before runtime, so that everything downstream — control, replay, evidence — has a declared baseline to work from.*

Organizations that create action manifests before scaling agent deployment will be better positioned to control runtime behavior, review sensitive actions, generate replayable evidence, and explain agent behavior later. Organizations that scale first will find themselves reconstructing the declaration after the fact — usually in response to a question they can no longer answer cleanly.

10. Open-Source Reference Implementation

Agent Action Manifest is published as a public, open-source reference implementation. The repository is public so teams can inspect the format, run the examples, adapt the schemas, validate manifests, and build compatible action-inventory workflows before making any platform commitments.

In practical terms, the repository includes:

- published JSON schemas defining the manifest format, so other systems can produce and consume manifests;
- a Python package with a clearly bounded public object model (built on Pydantic);
- a command-line interface for validating manifests, summarizing action inventories, and listing declared actions;
- example manifests for common enterprise scenarios — customer service, internal research, procurement, and finance workflows;
- a test suite run through GitHub Actions across supported Python versions;
- an Apache-2.0 license.

The implementation is intentionally small — a declaration format and a validator, meant to be readable and verifiable by an enterprise platform team, not a platform in itself.

11. Strategic Summary

Agent Action Manifest gives enterprises a practical way to document an AI agent's action surface before runtime deployment. It creates structure where agent capabilities are today scattered across prompts, tools, code, and informal assumptions. It helps teams declare tools, actions, authority requirements, review requirements, reliance requirements, payload policies, and redaction hints before an agent moves into governed operation — and it gives runtime control, replay, and evidence workflows a declared baseline to build on.

It does not claim to enforce anything, and it does not claim to be a governance platform. It claims something narrower and foundational: that governed deployment starts with a declaration, and that the declaration should be a structured, validated, reviewable artifact rather than an assumption.

Before enterprises can control what AI agents do, they need to declare what those agents may propose. That is the role of the Agent Action Manifest.