# Approval Intent Schema — Conceptual Design

## Purpose

The Approval Intent Schema exists to capture **human decisions** about agent behavior in a way that is explicit, auditable, and non-executable. It records *why* something was approved, *what scope* that approval applies to, and *what changes as a result* — without triggering execution or silently modifying authority.

An approval intent is a **signed statement of intent**, not an action.

---

## What an Approval Intent Is (and Is Not)

### An Approval Intent is:

- A deliberate human decision
- Explicitly scoped
- Deterministic in effect
- Auditable and reversible

### An Approval Intent is not:

- An execution
- A command
- A permission grant in code
- A hidden rule change

---

## The Two Approval Intents

The schema supports **two approval intents**, using a single language but different scopes.

### 1. Structural Approval (Policy-Level)

**Answers the question:**

> "Should this *kind of behavior* be allowed going forward?"

**Examples:** - Agents in Finance may auto-approve refunds up to €50 - This agent may write to production logs - Escalation to humans is no longer required for this action

**Characteristics:** - Applies broadly - Changes future eligibility - Affects multiple potential actions - Lives at the Organization, Domain, or Agent definition level

This is **OS-level intent**.

---

### 2. Situational Approval (Instance-Level)

**Answers the question:**

> "Should *this specific case* be allowed?"

**Examples:** - Approve this refund request - Allow this agent run to proceed - Override escalation for this incident only

**Characteristics:** - Applies once - Does not change future rules - Leaves structure untouched - Lives at the Runtime boundary

This is **execution-adjacent intent**, without execution.

---

## Why the Schema Encodes Intent, Not Mechanics

Encoding mechanics leads to: - Hidden permission creep - Unclear accountability - Accidental policy drift

Encoding **intent** ensures: - Auditability - Reversibility - Clear human accountability - Deterministic downstream behavior

Every approval intent must answer: 1. What is being approved? 2. How far does this approval reach? 3. What does this approval change — and what does it not change?

---

## Conceptual Schema Structure

### Core Identity

- **Intent ID**: Unique and immutable
- **Created By**: Human approver
- **Timestamp**: When the decision was made

### Scope

- **Approval Type**:
- STRUCTURAL (policy-level)

- SITUATIONAL (instance-level)

- **Target**:

- Organization
- Domain
- Agent
- Specific action attempt

**Reasoning**

- **Why this approval exists** (human-readable explanation)
- **What concern it resolves** (authority, escalation, identity mismatch)

**Effect**

- **What becomes allowed**
- **What remains restricted**
- **What explicitly does not change**

Every approval must clearly state:

> "This approval does not grant blanket permission."

---

## Non-Negotiable Constraints

These constraints are enforced by design: - An approval cannot execute anything - An approval cannot silently widen scope - An approval cannot downgrade future review requirements - An approval cannot self-repeat

Violating any of these would undermine safety and trust.

---

## How This Feels to Users

Users should experience approvals as: - "I'm approving this kind of behavior" - "I'm approving this one case"

They should never feel like they are: - Changing the system - Touching runtime internals - Granting permissions blindly

This distinction preserves clarity and trust.

---

## Roadmap Placement

- **Phase 4A**: Action staging infrastructure
- **Phase 4B**: Approval intent surfaced in UI
- **Phase 4C**: Persistence and audit of approval intent
- **Phase 5**: Runtime may reference approval intent (never the reverse)

---

## Design Doctrine

You do not have two approval systems.

You have **one approval language** with **two scopes**.

The schema exists to make misuse impossible by design.