Task Intelligence Truth Layer

Version 0.1

1. Purpose of This Document

This document serves as the **ground truth snapshot** for the Task Intelligence MVP.

Its goals are to:

- Establish a shared, canonical understanding of task completion semantics
- Freeze the current behavior of the evaluator and rule system
- Prevent semantic drift as the system evolves
- Enable fast onboarding of future contributors without re-litigating fundamentals

This document intentionally focuses on **truth and evaluation**, not UI, storage, or automation layers.

2. System Raison d'Être

The Task Intelligence system exists to answer one question reliably:

> **"Is this task complete and why?"**

It does so by:

- Treating real-world actions as immutable events
- Evaluating tasks deterministically against rulebooks
- Separating *truth evaluation* from *state mutation and automation*

This allows the platform to scale by shifting responsibility for correctness from people to code, while remaining explainable and auditable.

3. Core Primitives (Canonical Definitions)

Transaction

A transaction is the complete set of steps required by all stakeholders to successfully transfer ownership of a home.

Journey

A journey is the subset of steps within a transaction for which a single stakeholder has full or partial responsibility.

Task

A task is a discrete, named unit of work within a transaction or journey.

Tasks do not contain logic.
Tasks reference rulebooks to define completion.

Event

An event is an immutable record that a specific action or occurrence took place at a point in time.

Properties:

- Append-only
- Represents facts, not interpretations
- Timestamped in ISO 8601 UTC (`Z`)

Examples:

- `agreement.doc_987.received`
- `agreement.doc_987.executed`
- `inspection.scheduled`
- `buyer.opted_out`

Condition

A condition is an atomic, deterministic requirement that evaluates to true or false.

Conditions answer:

> **“What must be true?”**

Types (MVP):

- `occurred`
- `within`
- `all`
- `any`

- `not`

Conditions:

- Are reusable
- Contain no state
- Depend only on inputs (events + bindings)

Rulebook

A rulebook is a named set of one or more conditions that defines when a task is considered complete.

Rulebooks:

- Define completion semantics
- Contain logic, not state
- Are explainable
- Are versionable

Rulebooks answer:

> **“What does ‘done’ mean?”**

Rulebook Version

A rulebook version represents a specific, immutable definition of a rulebook applied at a point in time.

Versioning allows:

- Safe evolution of logic
- Auditability of historical decisions
- Deterministic replay

Evaluation

Evaluation is the deterministic, side-effect-free process of assessing whether a rulebook’s conditions are satisfied.

Evaluation:

- Produces `{ pass, reasons, evidence }`
- Never mutates state
- Never triggers automation

- Is fully explainable

Evaluation answers:

> **“Is this task complete and why?”**

Outcome (Post-Evaluation)

An outcome is an action taken *after* evaluation, based on the evaluated result.

Examples:

- Send notification
- Advance workflow
- Schedule reminder
- Escalate to concierge

Outcomes are explicitly **out of scope** for this MVP.

4. Binding & Reusability Model

Rulebooks are **templates**, not hard-coded instances.

They use **binding-aware atoms** to remain reusable:

```
{ ref: "docReceived" }
{ ref: "docExecuted" }
{ ref: "signingWindow" }
```

Bindings are provided at evaluation time:

```
{
  docReceived: "agreement.doc_987.received",
  docExecuted: "agreement.doc_987.executed",
  signingWindow: "P3D"
}
```

Missing bindings result in **deterministic evaluation failure**, not runtime exceptions.

5. Canonical Time Semantics (Locked)

The following semantics are **explicitly locked** by golden tests:

Anchor-based timing

For `within(event, anchor, duration)`:

- The anchor event establishes the clock start
- Only events **on or after the anchor timestamp** are considered
- The **first qualifying event** is selected
- Deadlines are **inclusive** (`<=`)

This correctly models statements like:

> “Sign within 3 days of receiving the agreement.”

6. Determinism Guarantees

The evaluator guarantees:

- Deterministic results for identical inputs
- Stable event selection rules
- No reliance on system clock
- No side effects
- No hidden state

This allows safe replay, auditing, and future automation.

7. Golden Tests (Non-Negotiable)

The following golden tests lock system behavior and must never be removed without explicit design review:

1. STANDARD_DOCUMENT_SIGNING happy path
2. Missing binding fails deterministically
3. Missing anchor fails deterministically
4. Inclusive deadline boundary
5. First event on/after anchor selection
6. Executed-before-anchor ignored
7. Composite `all` semantics
8. Alternative-path `any` semantics
9. Negation semantics (`not`)
10. UTC / midnight boundary integrity

These tests define **truth semantics**, not implementation details.

8. MVP Scope Boundaries

In Scope

- Condition evaluation
- Rulebook semantics
- Binding resolution
- Deterministic failure modes
- DB-free unit testing

Explicitly Out of Scope

- Persistence
- UI
- Notifications
- Scheduling
- Automation execution
- Permissions

These layers will be built **on top of** the truth layer, not inside it.

9. Immediate Next Steps (MVP → Iteration 2)

1. Add named condition groups (Domain-Specific Language expansion)
2. Improve evaluation evidence (selected event annotation)
3. Introduce materialized task-completed events
4. Integrate evaluator behind an API boundary
5. Begin mapping real platform tasks to rulebooks

10. Long-Term Vision

This truth layer enables:

- Automated task completion
- State-driven notifications
- Scalable concierge oversight
- Auditable workflows
- Safe incremental automation

By separating *truth* from *action*, the system can evolve without sacrificing reliability.