CODANCE PUBLICATION

How Aurora Core Executes an Assurance Method

A public technical overview of routes, bounded skills and reconstructible processing

Control	Value
Document	COD-PUB-CORE-001 v0.1 | 25 August 2026
Owner	Codance Ltd
Platform	Project Aurora working alpha
Purpose	Public technical architecture paper
Status	First substantive draft for competent review
Review discipline	Distinguish demonstrated evidence, engineering proposition and programme intent

PUBLIC TECHNICAL BOUNDARY

This paper explains the execution semantics required to make an assurance method attributable and reconstructible. It deliberately omits credentials, protected deployment details and security-sensitive configuration.

System boundary and design objective

Aurora Core is a campaign-blind processing environment. It does not accept campaign ownership claims embedded by a SUT and does not decide that a transmitter belongs to a campaign. Those responsibilities sit at the governed admission boundary and in Campaign Register. Core receives a canonical message only after admission and executes the currently published route associated with its resolved iDevice.

The design objective is not simply reliable orchestration. It is to preserve an evidential account of which route operated, which work was scheduled, which products each step consumed and emitted, how failures were disposed, and which terminal outputs belong to the originating input.

Plane	Primary responsibility	Must not do
Admission / Register	Authorise, attribute and record governed transmissions.	Trust SUT-supplied campaign UUIDs.
Core processing	Execute published routes and preserve work/product lineage.	Invent campaign ownership or rewrite completed history.
Device / skill runtime	Perform one bounded step under declared inputs and configuration.	Silently acquire undeclared evidence or dependencies.
Output and retention	Publish terminal products and preserve retrievable evidence.	Detach a report from its source Processing Episode.

1. Canonical input and episode creation

Input Router normalises a governed delivery and resolves actor_key, space_key and channel_key against the active route registration. The resolved iDevice provides the entry stage, DAG mode and actor policy. Campaign admission records the registered transmitter, Transmission Definition, collection point, execution attribution and gate state. Only after these checks does Core persist the inbound stm_message and create the Processing Episode.

- msg_id is the canonical identity of the persisted input or output.
- run_id groups processing activity associated with an input.
- dag_run_id identifies the dependency-managed execution of the published plan.
- base_msg_id anchors the DAG to its initiating canonical message.
- plan_version_text identifies the route definition used for the episode.

IDENTITY RULE

A Signal is not identified by content similarity. It is the canonical persisted form of one governed Transmission and carries the lineage supplied by admission and routing.

2. Published routes and immutable history

A route definition contains iDevices, stages, step definitions, dependencies, products and device bindings. Publication creates a versioned plan. New Processing Episodes use the active published version; existing work retains the version under which it was created. This ensures that later configuration changes do not retroactively alter the meaning of completed evidence.

A stage can contain steps and sub-stages. Steps declare hard dependencies through requires and may add stage-specific optional dependencies. DAG construction resolves these definitions into executable nodes and edges. A node becomes claimable only when required predecessors have completed successfully and its load and weight constraints permit execution.

Definition	Runtime evidence
iDevice → entry stage	Resolved entry_stage and plan_version in message metadata
Stage → steps / sub-stages	Plan tree retained for the Processing Episode
Step → requires / products	DAG edges and typed products
Step → skill_id	Device binding and invoked package identity
Published version	Version text/hash associated with run and work records

3. Work-in-progress records

Every step invocation is represented as work rather than an invisible function call. The work record contains the current canonical message, trace metadata and the material products available from prior steps. This allows a later step to consume declared earlier results without flattening them into an untraceable prompt.

The runtime reserves work, binds it to a DAG step and records claims, attempts, completion sequence, products and errors. A completed step remains historical evidence. If the method authorises adaptive work, the system may mutate future uncompleted structure through governed functions, but it must not rewrite a completed node or erase the reason for the mutation.

State question	Evidence retained
Why was the step eligible?	Satisfied dependencies and DAG edge state
What did it receive?	Input message, cognition metadata and selected prior products
What executed?	Step identity, device binding, skill package and attempt
What happened?	Completion, failure, skip or blocked disposition
What did it produce?	Typed product identity, value or retained reference
What changed later work?	Governed mutation record and origin step

4. Bounded skill execution

A skill is the installed implementation of one step on a device class. Its package contains executable code, configuration and requirements. Stable package and step identities allow behaviour to evolve through explicit versions without breaking route references. Material semantic changes are recorded in configuration and source; a rename is a governance operation because it changes step-to-skill and installed-package identity.

Skills validate their expected input products, perform the narrow task and emit a typed result. They may call deterministic algorithms, local services, controlled models or registered data functions, but only within the method and deployment boundary. A skill failure must be represented as a failure product or terminal error path; absence of an output is not valid completion.

- Validate object kind, schema version and required source step.
- Verify hashes, identities and commissioned configuration where relevant.
- Reject ambiguous multiplicity where exactly one product is required.
- Retain material request and response bodies or hashes according to policy.
- Emit typed products with version, provenance and audit events.
- Never convert an exception into a favourable result or silently continue.

5. Deterministic controls and model calls

Core is capable of dispatching work that uses a model, but route control and evidential integrity do not depend on a model's compliance. Deterministic code validates identities, schemas, hashes, boundaries, gate state, idempotency and product structure. A model is invoked only by a commissioned skill and remains an attributable influence with a specific endpoint, model name, digest, prompt guide and request material.

The ClearLedger demonstration pinned qwen3.5:4b by digest and retained the request material generated under each evidence guide. The model identity remained invariant while the configured guide altered the request. This is precisely the kind of method influence that would be obscured if model calls were treated as an unstructured black box.

DESIGN LAW

Probabilistic capability may contribute to an assessment; it does not control the deterministic boundary that establishes what evidence entered, which code ran or whether a product is structurally valid.

6. Multiple outputs, derived inputs and terminality

One input may produce multiple outputs. An output can communicate an intermediate result, trigger a separately governed input, or provide the final response to the initiating request. Parent and input identities preserve those relationships. The initiating sequence remains open until an output explicitly carries is_final=true.

This matters for assurance campaigns because an observed terminal SUT event may generate a closure request while its own sensor-ingest route still completes normally. The derived closure input begins a new Processing Episode; it does not 'continue' the earlier input. Each episode has its own route, steps and terminal disposition, linked through message and transmission lineage.

Pattern	Meaning
Intermediate output, is_final=false	The current input sequence remains open.
Terminal output, is_final=true	The current input sequence is closed.
Output used as new input	A new governed Processing Episode begins.
Parent-linked report	The report remains associated with its initiating request.
Failure terminal output	Processing closes with an attributable error disposition.

7. Failure semantics

Failure location determines evidence meaning. Admission refusal occurs before Core and is recorded as a Transmission Failure. A skill exception, malformed product or unsatisfied dependency after ingress is a Processing Failure. The DAG reconciler marks dependent work as blocked or skipped and preserves the failing step and error rather than presenting an incomplete route as successful.

Failure class	Required disposition
Address or definition mismatch	Register refusal with failure code and ledger identity
Gate closed / transmitter unauthorised	Pre-Core Transmission Failure
Skill validation or execution exception	Failed DAG step with typed error evidence
Dependency failed	Dependent step skipped as blocked_by_failed_step
Required evidence absent	Refusal, failure or UNKNOWN according to the method
Terminal report cannot be emitted	Processing Episode remains failed, never silently complete

8. Reconstruction from report to source

The execution record supports reverse review. A reviewer can begin with a terminal report msg_id, identify its run and source step, inspect the DAG run and every step outcome, retrieve the assessment and evidence-snapshot products, follow root transmission and SUT execution identities, and examine the registered sensor acquisition and source journal records.

In the ClearLedger pair-reporting route, the human request and internal comparison were deliberately separate episodes. The request route validated the pair selection, dispatched an idempotent internal comparison request and emitted an acceptance receipt.

The comparison route gathered both registered execution reports, assessed the controlled invariants and difference, and emitted the human-readable report. All six steps completed, leaving two distinct but linked histories.

PUBLIC LIMITATION

This overview states architectural semantics, not a security configuration or formal verification. Production qualification must separately address access control, secrets, isolation, resilience, availability and adversarial operation.

Document control and review

This is a first substantive public draft prepared for structured review. Before release, Codance will verify technical identities and claims against the controlled evidence, resolve reviewer comments, assign the approved publication version, regenerate the PDF and record the final file SHA-256.

Review question	Required disposition
Is every statement clearly classed as demonstrated fact, engineering proposition or programme intent?	Confirm, narrow, source or remove.
Can a competent reader identify the controlled boundary and prohibited over-claims?	Make the boundary explicit.
Do technical identities and results match retained evidence?	Verify against controlled records.
Does the document disclose material limitations and credible contrary interpretations?	Add or strengthen limitations.
Is the document useful without access to confidential implementation detail?	Improve explanation or supporting public evidence.