SALESFUSION

Proof-of-Run for Recurring Agents

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Proof-of-Run for Recurring Agents

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	proof-of-run-for-recurring-agents
Canonical path	/technical-papers/proof-of-run-for-recurring-agents
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex needs proof that recurring revenue work actually ran, not just that an agent or cron job started. Wake-up lists, signal scans, pipeline reviews, follow-up routers, and reporting loops need run receipts, artifacts, no-result handling, and verification before they count as operating evidence.

Evidence-status box

> SalesFusion uses proof-of-run as an operating pattern for recurring agent work. The pattern records schedule events, starts, exits, artifacts, verification results, blockers, failures, no-result states, retries, reconciliation, and rollback. The design addresses the failure class in which a scheduled job or process exit does not establish useful completion. This paper does not report incident rates or claim comparative reliability improvement.

> The design uses observability concepts consistent with the OpenTelemetry Specification and append-only history concepts. It also follows the implementation principle that a successful HTTP or process response does not prove downstream completion.

Abstract

Recurring agents are easy to schedule and hard to prove. A cron trigger can fire, a worker can start, a process can exit with success, and an agent can report that it is done while the intended artifact is missing, invalid, stale, duplicated, or unusable. For business workflows, assignment and exit are not completion. Proof-of-run requires a state model that distinguishes scheduled, started, exited, output-created, verified-complete, blocked, failed, and no-result states.

This paper defines a reliability protocol for recurring agents. Paper 13 is its companion at the outcome layer: proof-of-run establishes execution evidence, while an outcome ledger records what happened after execution without asserting causation. The protocol specifies expected artifacts, idempotency keys, action receipts, traces, reconciliation checks, retry rules, blocker classification, and rollback behavior. It treats no_result as a valid terminal state when the agent ran correctly but no qualifying work existed. It treats blocked as distinct from failed when the run could not proceed because of missing evidence, authority, dependency, or external system state. SalesFusion uses this pattern and proposes an evaluation suite for false completion and incident-rate comparison.

The problem

Recurring agents create a special kind of ambiguity. They run on a schedule, not always in response to a human watching the work. If they fail silently, the system may assume ongoing coverage that does not exist. If they retry without idempotency, they may duplicate consequential work. If they report success without artifact verification, operators cannot distinguish "nothing needed to happen" from "the agent did not check" or "the check ran but produced unusable output."

The common status vocabulary is too small: pending, running, success, failed. It hides the difference between a schedule event and a worker start. It hides process exit from business completion. It hides the difference between no qualifying input and a broken query. It hides whether an external action was attempted once, twice, or not at all.

AI agents increase this risk because they can produce plausible narrative status. A message saying "completed" may be only a self-report. It is not an artifact, not a connector receipt, not a reconciliation result, and not a verified run state.

Contributions

This paper contributes:

- A proof-of-run state model for recurring agents.
- A required artifact contract for verifying completion.
- A distinction between blocked, failed, and no_result.
- Idempotency, reconciliation, retry, and rollback rules for recurring work.
- An implementation pattern using run records, event envelopes, traces, metrics, logs, artifacts, and receipts.

- An evaluation design for false completion, duplicate work, blocker quality, and recovery behavior.

Core definitions

Scheduled means a trigger or scheduler created an expected run. It does not prove a worker started.

Started means a worker accepted the run with a workflow version, input digest, idempotency key, and trace identifier.

Exited means the worker process returned. Exit can be successful, failed, timed out, or cancelled. It does not prove expected artifacts exist.

Output-created means the run produced an artifact or action receipt matching a known output type. It may still be invalid.

Verified-complete means the expected artifact contract passed deterministic checks and any required reconciliation.

Blocked means the run could not proceed because a dependency, authority, source, connector, policy, or human input was missing or invalid.

Failed means the run attempted work and encountered an error that prevented completion.

No-result means the run executed the required checks and correctly found no qualifying work or no qualifying output for the period.

Proof-of-run architecture

The protocol separates scheduling and orchestration events from evidence of useful work.

State	Evidence required	Typical next state
scheduled	Schedule event with expected run window	started, missed, cancelled
started	Worker claim with workflow version and trace ID	exited, blocked, failed
exited	Process exit code, duration, logs	output_created, failed, verification_failed
output_created	Artifact reference or action receipt	verified_complete, verification_failed
verified_complete	Artifact contract passed, reconciliation passed	terminal or outcome review
blocked	Blocker class and required resolution	resumed, cancelled, expired
failed	Error class, retry eligibility, trace	retry, rollback, cancelled
no_result	Query/check evidence proving no qualifying work	terminal

The minimal run record includes:

```
{
  "run_id": "run_001",
  "workflow_id": "wf_daily_review",
  "workflow_version": "wf_v12",
  "tenant_id": "tenant_ref",
  "scheduled_for": "2026-07-16T09:00:00Z",
  "input_digest": "sha256:...",
  "idempotency_key": "tenant:workflow:window",
  "trace_id": "trace_001",
  "expected_artifacts": ["review_packet", "run_summary"],
  "status": "started"
}
```

The artifact contract is specific to the workflow. A reporting run may require a dated report, source query digest, reconciliation totals, and validation result. A monitoring run may require a checked source list, qualifying event count, no-result proof when empty, and alert packet when non-empty. A mutating run may require an action receipt from the external system and an internal reconciliation event.

State transitions and decision rules

Proof-of-run is strict about transition meaning:

```
scheduled -> started -> exited -> output_created -> verified_complete
                    \-> blocked -> resumed -> exited
                    \-> failed -> retry -> started
                    \-> no_result
```

no_result should not be treated as failure. It is a successful negative result when the workflow's expected job is to check for qualifying work. But it must be evidenced. A blank output is not enough. The run must record the query or check performed, the source set, the time window, and the criteria that produced zero qualifying records.

Retries must be idempotent. A retry can re-run analysis, but a retry of a consequential external action must either return the prior receipt or prove no prior action occurred. A reused idempotency key with a different request body should be rejected.

Rollback depends on action class. Internal reversible actions can often be rolled back by restoring prior state or superseding a record. External consequential actions may not be fully reversible. The run record must distinguish rollback performed, rollback unavailable, and rollback requires human escalation.

Implementation

SalesFusion uses the proof-of-run pattern with run ledgers, expected-artifact checklists, blocker states, and reconciliation checks. The runtime captures schedule event, start event, process exit, trace identifier, logs, metrics, artifacts, action receipts, verification result, and outcome review as separate records. OpenTelemetry-compatible traces, metrics, and logs provide the observability vocabulary; the domain ledger provides the business state.

The implementation pattern uses these interfaces:

```
POST /workflows/{workflow_id}/runs
GET  /runs/{run_id}
POST /runs/{run_id}/heartbeat
POST /runs/{run_id}/artifact
POST /runs/{run_id}/block
POST /runs/{run_id}/fail
POST /runs/{run_id}/verify
POST /runs/{run_id}/retry
POST /runs/{run_id}/rollback
```

Verification is deterministic where possible. A run cannot mark itself verified by writing prose. The verifier checks artifact existence, schema validity, source window, required fields, action receipts, duplicate detection, and reconciliation rules. For recurring runs, the verifier also checks that the expected schedule window is covered and not double-counted.

Blockers are typed:

Blocker class	Example condition
source_unavailable	Required source could not be read.
authority_missing	Consequential action lacks valid approval.
evidence_insufficient	Required source or claim is missing, stale, or disputed.
connector_ambiguous	External system response cannot prove action state.
rate_limited	Connector or policy requires delay.
human_input_required	Workflow needs a decision or correction.

This classification matters because retrying an authority blocker is not the same as retrying a transient connector timeout.

Reconciliation, retries, and rollback

Reconciliation answers: did the world end up in the state the run believes it created? For internal artifacts, reconciliation can compare artifact digest, schema, and ledger entry. For external actions, it can compare connector receipts, external object identifiers, timestamps, and idempotency keys. For reporting, it can compare source totals and deterministic aggregations.

Retry rules:

```
if blocker is authority_missing:
  do not retry automatically
if error is transient_connector_failure:
  retry within capped policy using same idempotency key
if prior action receipt exists:
  return prior receipt and mark duplicate suppressed
if request body changed under same idempotency key:
  reject and audit
if repeated verification failure:
  stop retries and escalate
```

Rollback rules should be declared before execution. A workflow that cannot roll back a consequential action should require stricter approval. A rollback event should reference the original run, action receipt, rollback method, result, and remaining residual risk.

Evaluation

The immediate evaluation is a reliability benchmark, not a business outcome study. It should include scenarios where a schedule fires but the worker never starts, worker starts but exits before artifact creation, process exits successfully with invalid artifact, connector returns ambiguous response, retry would duplicate an action, no qualifying input exists, required authority is missing, and rollback is unavailable.

Metrics:

Metric	Definition
Verified-completion accuracy	Runs marked complete only when artifact contract passes.
False-completion rate	Runs marked complete despite missing or invalid evidence.
Missed-run detection	Scheduled but unstarted runs are detected.
No-result correctness	Empty periods are correctly distinguished from failures.
Duplicate-work rate	Retries do not duplicate consequential actions.
Blocker classification quality	Blockers are assigned actionable classes.
Retry recovery	Eligible transient failures recover without violating idempotency.
Rollback proof	Rollback records show what was reverted and what remains.

Incident-rate comparison requires a baseline system and a defined operating window. Until that exists, the proper claim is that the protocol is implemented and testable.

Risks and failure modes

False completion is the primary risk. It can come from trusting process exit, trusting agent self-report, accepting empty output, or skipping reconciliation. Duplicate consequential action is another serious risk, especially when

retries occur after timeouts or ambiguous connector responses. Silent missed runs are also dangerous because recurring work can appear to be covered while no worker is actually running.

Other risks include log loss, artifact tampering, cost runaway from retry loops, stale schedule windows, and rollback theater where a system records rollback without proving reversion. Controls include append-only events, integrity hashes where needed, idempotency, capped retries, dead-letter queues, independent verification, and human escalation for irreversible actions.

Limitations

Proof-of-run does not prove that the agent chose the best action. It only proves that a defined run occurred, produced or did not produce expected artifacts, and passed or failed verification. It depends on correct artifact contracts. A bad contract can verify the wrong thing. It also adds latency and storage overhead.

SalesFusion's current evidence supports use of an operating pattern that addresses the described failure classes. It does not establish their frequency and does not support quantified claims about reduced incidents, increased reliability, or improved business outcomes.

Practical implications

Recurring agents should be operated like auditable jobs, not like chat sessions. Each run needs a schedule record, start record, trace, artifact contract, verification result, blocker or failure class, and reconciliation result. Operators should review blocked and no-result states, not only failures.

Product teams should remove success as a standalone terminal state. Use verified_complete, no_result, blocked, failed, cancelled, and rolled_back instead. A process exit code can be useful telemetry, but it is not proof of business completion.

Conclusion

Proof-of-run is the evidence layer recurring agents need before their outputs can be treated as operationally verified. It separates scheduled, started, exited, output-created, verified-complete, blocked, failed, and no-result states; requires expected artifacts and reconciliation; and treats idempotency, retries, and rollback as first-class controls. SalesFusion uses this pattern and proposes reliability tests for false completion, duplicate work, missed runs, blocker classification, and rollback behavior.

References

- OpenTelemetry, "OpenTelemetry Specification." https://opentelemetry.io/docs/specs/otel/
- IETF, "RFC 9110: HTTP Semantics." https://www.rfc-editor.org/rfc/rfc9110.html

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.