DOCUMENTATION PAGE

# First-Task Guide

# First-Task Guide

This is a practical first task for a new auto-orch operator working on a candidate-evidence handoff. It is a product-side walkthrough: inspect the declared contract, run the canonical read-only preflight against the supplied sample, preserve the exact result, and stop at the candidate-only boundary. The preflight does not grant authority, approve a candidate, or prove a platform-owned transition.

## Candidate Evidence Chain Walkthrough

Follow the evidence chain in order. First locate the human-readable contract at `docs/candidate-evidence-preflight-contract.md`, then compare its machine-readable candidate-handoff declarations in `journeys/candidate-evidence-handoff-manifest.json` and this page's `content/first-task-guide-reader-journeys.json`, alongside the model, executable tool, and sample evidence. The contract explains the required fields and exact command/name matching; `models/candidate_evidence_handoff_contract.json` carries the synchronized shape; `tools/candidate_evidence_preflight.py` checks declarations, supplied observations, and the independently supplied review verdict; and `tests/fixtures/candidate_evidence/valid.json` is the sample candidate to read. Keep declarations (what a journey says should have happened) separate from evidence (what the supplied record says was observed).

The chain ends in a product conclusion: the supplied record either satisfies the read-only preflight or is rejected with discrepancies. The tool does not execute commands claimed inside the evidence and does not write a result artifact. A pass is therefore evidence that this supplied handoff is contract-complete, not evidence of approval, publication, promotion, scheduling, rollback, release-pointer creation, activation, or a live-directory change.

# Before You Run the Preflight

Open the contract before opening the sample. Confirm its synchronized machine-readable companion, the model, the preflight tool, and the valid fixture; then note the required independent review path `code-reviews/content-review.verdict.json`. A named review path is a required location, not proof that a review artifact exists. Also keep the sealed authority and registered source boundary in view: `sources/authority.json` and `registries/sources.json` are read-only product inputs, and the guide's three registered source references must not be supplemented with guessed provenance.

Do not edit the valid fixture to make it agree with a declaration, do not rewrite the contract or model while investigating, and do not treat a fixture's `commands_run` entries as commands that this preflight will replay. Capture the input path and the exact command you use. If a declaration and an observed field disagree, retain both values so a later reader can see whether the discrepancy belongs to the manifest, the evidence record, or the review verdict.

# Run the Canonical Preflight

From the repository root, run the canonical read-only preflight against the valid sample exactly as follows. Pin the candidate-handoff manifest and contract explicitly so the invocation cannot silently select an unrelated journey manifest:

```
PYTHONDONTWRITEBYTECODE=1 python3 -B tools/candidate_evidence_preflight.py --
```

The tool reads the selected manifest, contract, and sample, checks the required journey coverage, exact names, traces, allowlisted commands, integer exit codes, observed-result fields, and review-verdict path, and returns exit code `0` with a JSON `passed: true` and `verdict: "pass"` when all supplied facts agree. A non-JSON invocation reports the equivalent `PASS: candidate-evidence preflight passed` line. If the selected manifest is stale or conflicts with the pinned handoff, the command returns exit code `1` and emits the exact declaration or evidence discrepancies; that rejection is the result to preserve. It is read-only: it does not run the commands named in the sample, edit the sample, alter authority, change a validator, or create a publication pointer. Preserve the stdout, stderr, exit code, input path, and any machine-readable error list as the run evidence.

The observed attempt for this walkthrough rejected the supplied declarations: it reported `manifest.required_review_verdict_path`, a missing `manifest.evidence_schema`, a missing `manifest.review_verdict`, and missing `required` and `expected_exit_code` fields on all three candidate-handoff journeys. Keep that rejection beside the sample and route it as declaration discrepancy evidence. Do not turn it into a pass by editing authority, the fixture, the validator, or the independent review evidence; a later governed manifest or review result may supplement the record but cannot erase this observation.

## Interpret Passing and Failing Results

A passing result means the supplied record is structurally and declaratively consistent with the canonical handoff contract: the pinned journeys are covered, their names and traces match, each claimed command is an exact allowlist member, each claimed exit code has the declared type and value, the observed-result fields are present, and the required review verdict is readable and acceptable. It does not mean that the sample's claimed commands ran during this invocation, that a product conclusion is a platform attestation, or that any transition is authorized.

A rejection means the tool found a discrepancy. Declaration discrepancies can include a changed journey name, missing acceptance coverage, an unallowlisted or whitespace-altered command, a changed expected exit code, or a manifest shape that no longer matches the contract. Evidence discrepancies can include missing observed fields, incomplete journey records, the wrong review path, or a supplied status other than `passed`. Read the exact error and classify it against the original files; do not normalize it away. A rejected sample or conflicting declaration is evidence to preserve and escalate, not a reason to edit authority, fixtures, validators, or review evidence.

## Escalate Discrepancies

Escalate with the smallest reproducible packet: the contract and manifest paths, candidate input path, exact command, exit code, stdout and stderr, the tool's exact rejection lines, the affected journey or field, and the source references that establish the expected boundary. State whether the conflict is in a declaration, the supplied candidate evidence, or the independent review record. Keep the original rejected record beside any later corrected or rerun record; a later result supplements history and does not erase the first observation.

Route the discrepancy through the owner named by the governing process. If authority or a registered source conflicts with the candidate, preserve the sealed value and report the conflict for authority review. If the evidence is incomplete, request a new governed evidence record rather than filling fields from assumptions. If the independent review artifact is missing or malformed, leave handoff completion unresolved. Do not repair a failed result by editing authority, fixtures, validators, or review evidence, and do not run an unallowlisted command as an unofficial substitute for the canonical check.

## Candidate-Only Boundary

Finish by confirming what did not happen. The preflight read the contract, manifest, candidate sample, and required review path; it did not approve, activate, promote, publish, schedule, roll back, create or change a release pointer, or write a live directory. A passing result remains a product-side candidate-evidence conclusion, and a rejection remains preserved evidence for escalation. Neither result changes the sealed authority or turns a candidate into a release. If a separate deterministic projection is later generated, `build/index.json` is still generated product output, not a publication pointer or a transition attestation.

Before closing the task, record the preflight result and verify that no non-candidate transition occurred: no authority or fixture was rewritten, no validator or review record was altered, no publication boundary was crossed, and no live-directory or release-pointer action was taken. The correct ending is a synchronized, inspectable candidate handoff with its pass or rejection evidence intact and any unresolved discrepancy explicitly escalated.

---

**Provenance:** registry-listed (no candidate packet)
**Registry Source:** `content/first-task-guide.md`