DOCUMENTATION PAGE

Candidate Evidence Handoff Preflight

Candidate Evidence Handoff Preflight

This page is a bounded troubleshooting guide for auto-orch operators and maintainers preparing a candidate-only evidence handoff. It keeps product conclusions, source authority, reader evidence, and platform-owned attestations distinct. A failed observation is still evidence: preserve its exact command, exit code, output, and owner rather than rewriting it as success.

Preflight Failure Triage

Start with the complete read-only result, not with a guessed repair. Preserve the evidence path, every error string, the first observable failure, and the declared manifest and contract paths. `run_preflight()` in `tools/candidate_evidence_preflight.py` reads those inputs and the independent review verdict; it does not execute a claimed command or turn a declaration into proof. Use `docs/candidate-evidence-preflight-contract.md`, the executable validator, `sources/authority.json`, and `registries/sources.json` as the source-linked diagnosis boundary. Classify the result as schema/evidence shape, command allowlist, journey identity/trace, or fixture/review-path failure, then repair only supplied candidate evidence or route a declaration problem to its owner.

The deterministic focused check is `python3 -m pytest -p no:cacheprovider tests/test_candidate_evidence_preflight.py`; compare the reported case with the matching fixture in `tests/fixtures/candidate_evidence/`. A known declaration-drift symptom is a contract that describes five checks and JSON-tool commands while the executable validator, model, and candidate fixtures pin three journeys and the exact `-B` pytest/smoke commands. That conflict is itself evidence: do not choose a side or hide it in this page. If an observation cannot be tied to an authoritative source, exact journey, and reproducible command, retain it as `unresolved` and stop rather than borrowing a nearby result.

Schema and Evidence Shape Failures

Observable symptom: The preflight reports `schema_version`, `kind`, `journeys`, a missing required field, `observed_result`, an invalid status, or coverage errors. A common example is `tests/fixtures/candidate_evidence/malformed-evidence.json`, whose first command claim has an exit code but no observed-result field.

Authoritative diagnosis: Read the required top-level, journey, and command claim fields in `docs/candidate-evidence-preflight-contract.md`, then confirm the executable checks in `validate_evidence()` and the schema constants in `tools/candidate_evidence_preflight.py` (also mirrored by `models/candidate_evidence_handoff_contract.json`). The record needs schema version 1, kind `candidate-evidence-preflight-evidence`, all declared journeys, exact names, non-empty `steps_taken`, `source_refs`, and `traces_to`, a string-list `blockers`, and command claims with an integer `exit_code` and non-empty `observed_result`. The validator requires each evidence record's status to be `passed`; that says the record is structurally complete for preflight, not that every claimed command exited zero. The pinned smoke claim therefore preserves its expected exit code 1 and its failed observation.

Safe remediation: Rebuild the record from the actual run and add only observed values, retaining failures in `observed_result` and `blockers`. Populate source references from the declared artifact, not from memory. Do not loosen the schema, delete a failed journey, replace an empty field with plausible prose, or change the validator to accept malformed evidence.

Deterministic confirmation: Run `python3 -m pytest -p no:cacheprovider tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_malformed_evidence_is_rejected` against `tests/fixtures/candidate_evidence/malformed-evidence.json`, whose first command claim omits `observed_result`; use `tests/fixtures/candidate_evidence/valid.json` for the complete shape. If the real record still lacks an authoritative observation, stop with that field unresolved and escalate to the evidence owner; do not fill it with a summary of what the command was expected to do.

Command-Allowlist Failures

Observable symptom: The error includes `command allowlist`, or a command looks almost right but differs by a leading/trailing space, doubled space, newline, or a different executable. `tests/fixtures/candidate_evidence/disallowed-command.json` deliberately claims `python3 tools/validate_content.py` for a journey that is pinned to a different command.

Authoritative diagnosis: The command allowlist and each journey's exact command are pinned by `tools/candidate_evidence_preflight.py`, the candidate model, and the focused tests. For the executable candidate-handoff route, the only two exact allowlist strings are `python3 -B tests/check_documentation_smoke.py` and `python3 -B -m pytest -p no:cacheprovider tests/test_product_contract.py`. The first two pinned journeys claim the pytest command with exit code 0; the documentation-validation journey claims the smoke command with exit code 1. The test `test_command_claims_are_not_normalized_before_allowlist_matching` confirms that whitespace is not silently normalized, and a claim must also equal its journey's declared command. The human-readable contract currently contains a different JSON-tool allowlist; that declaration conflict must be escalated, not papered over by this guide.

Safe remediation: Copy the exact allowlisted string into the matching journey claim and preserve the command's separately observed exit code and output. If the needed check requires another route, do not add it to the candidate record or alter the allowlist as a page repair; ask the owner of the manifest/contract for a governed change. A command being useful or successful does not make it evidence for a different journey.

Deterministic confirmation: Run `python3 -m pytest -p no:cacheprovider tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_disallowed_command_is_rejected tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_command_claims_are_not_normalized_bef` against `tests/fixtures/candidate_evidence/disallowed-command.json`. If the exact command cannot be run in the permitted route, preserve the failure and stop; never substitute a locally convenient command, strip whitespace for a claim, or report that the allowlist was satisfied.

Journey Identity and Trace Failures

Observable symptom: The result reports a journey `.name`, duplicate or missing `journey_id`, journey coverage, `traces_to`, acceptance mapping, or pinned handoff mismatch. `tests/fixtures/candidate_evidence/invalid-journey-name.json` shows the typical case: the ID remains present while the name silently drops required wording.

Authoritative diagnosis: The pinned IDs, names, position-specific traces, commands, and expected exit codes are the constants in `tools/candidate_evidence_preflight.py`. For this handoff they cover exactly AC-1, AC-2, and AC-3, in order, with traces `["AC-1"]`, `["AC-2"]`, and `["AC-3"]`. The expected IDs are `journey.candidate-evidence-handoff-preflight`, `journey.candidate-evidence-handoff-template`, and `journey.candidate-evidence-handoff-documentation-validation`; their names must be copied byte-for-byte from the pinned declarations and fixtures. The contract's exact-name and coverage rules explain why an identifier, shortened name, reordered trace, or extra journey is not equivalent. Compare `journeys/user_journeys_manifest.json` with the pinned candidate manifest, model, tool, and fixture; do not silently reconcile a mismatch by editing this page.

Safe remediation: Recreate the evidence entry from the current pinned manifest, copying the journey ID and name byte-for-byte and retaining the declared `traces_to` value. Keep a failed observation attached to its original journey; do not move it to a journey that happens to pass. If the manifest, contract, and executable expectations disagree, preserve all three facts and escalate the declaration conflict rather than changing a registry or source reference in this page.

Deterministic confirmation: Run `python3 -m pytest -p no:cacheprovider tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_invalid_journey_name_is_rejected tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_journey_acceptance_mapping_is_pinned_` against `tests/fixtures/candidate_evidence/invalid-journey-name.json`. If the ID, exact name, or trace cannot be established from synchronized declarations, stop checklist completion and escalate to the manifest/contract owner. A successful check on the wrong journey is not coverage, and a current publication-reader manifest is not evidence for this candidate handoff.

Fixture and Review-Path Failures

Observable symptom: A fixture is missing required evidence, the evidence does not name `code-reviews/content-review.verdict.json`, the verdict file is absent or unreadable, or its JSON has the wrong schema, verdict, or forbidden finding severity. `missing-review-path.json` tests the evidence-side

path omission; `run_preflight()` separately reads the required file from the supplied repository root.

Authoritative diagnosis: The review-verdict requirement and its exact relative path are defined in `docs/candidate-evidence-preflight-contract.md`. `validate_review_verdict()` in `tools/candidate_evidence_preflight.py` requires schema version 1, verdict `pass` or `clean`, an array of findings, and no High or Critical finding. `run_preflight()` resolves the required path under the supplied `--root` and reads it independently. The valid fixture is a shape example, not proof that the independent artifact exists in the current run; a named path is not the same as a readable verdict. The `missing-review-path.json` fixture tests the evidence-side path omission, while the CLI path check tests a separately supplied verdict file.

Safe remediation: Use the valid fixture to check field shape, then supply the independently produced verdict at the exact path under the preflight root. Keep a missing, stale, malformed, or unverified verdict as a blocker; do not fabricate a verdict, copy one from another handoff, or treat a local content validator result as independent review. Preserve any failed fixture or review read as evidence for the owner who can provide the artifact.

Deterministic confirmation: Run `python3 -m pytest -p no:cacheprovider tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_missing_review_path_is_rejected tests/test_candidate_evidence_preflight.py::CandidateEvidencePreflightTests::test_run_preflight_requires_a_readable_inc` and compare the record with `tests/fixtures/candidate_evidence/valid.json`. For a controlled CLI replay, use the test's read-only shape: `python3 tools/candidate_evidence_preflight.py --evidence <candidate-evidence.json> --manifest journeys/user_journeys_manifest.json --contract docs/candidate-evidence-preflight-contract.md --root <preflight-root> --json`. If the exact review artifact is unavailable, malformed, or not independent, or fixture and manifest declarations diverge, stop and escalate; a preflight cannot self-certify independent review.

Escalation Boundary

Stop and escalate when an authority, source reference, exact journey identity, allowlisted command, observed result, fixture, or independent review artifact is missing or contradictory; when a command needs a route outside the pinned allowlist; or when a failed observation could be interpreted as a product defect without authoritative recurrence and impact evidence. Failed observations remain evidence and must be carried forward verbatim with their owner and blocker. A disagreement between the human-readable contract, manifest, model, executable constants, or focused fixtures is a declaration failure, not permission to edit whichever file makes the preflight pass. This page may describe a boundary, but it cannot invent provenance, retry counts, affected claims, platform attestations, approval, or a repair target.

The handoff remains candidate-only. Neither this guide nor a preflight pass approves, promotes, publishes, schedules, activates, rolls back, changes a release pointer, or writes a live directory. Those transitions and platform-owned attestations belong to their governing authorities. When in doubt, retain the candidate, record the deterministic failure, and route the unresolved question to the authority named by `sources/authority.json` and `registries/sources.json` rather than making the evidence appear complete.

Provenance: registry-listed (no candidate packet)
Registry Source: `content/candidate-evidence-handoff-preflight.md`