DOCUMENTATION PAGE

Governed Execution Engine

Governed Execution Engine

The Governed Execution Engine documentation is maintained as one semantic source and evidence model. Reading modes may change emphasis and presentation, but they do not create competing versions of a platform claim.

This overview keeps the documented claim aligned with its source references and evidence model. It describes the documentation structure for public-safe reading modes, so the same claim can be read with executive, engineering, or AI emphasis without adding a separate platform claim or unsupported outcome.

How to read this page

Read the page as one traceable explanation. Executive mode highlights the governing idea; engineer mode makes the source and evidence relationships explicit; and AI mode exposes that same structure for consistent interpretation. These are reading modes over one semantic source and evidence model, not separate claims or competing documents.

Start here

If you have no repository context, begin here. First locate the governing source boundary in `sources/authority.json`, then open `registries/sources.json` and match `source.naming-decision`, `source.documentation-boundary`, and `source.platform-delivery-contract` to their locators, revisions, digests, and `public-safe` disclosures. Next read the `Semantic source`, `Projection`, and `Release boundary` sections of `architecture.md` to understand how `content/`, `registries/`, and `models/` relate to authored Markdown and the deterministic `build/index.json` projection. The `source.platform-delivery-contract` record is the contract governing this delivery profile's reproducibility; the reproduced evidence is `build/index.json` together with the deterministic content-validation and projection checks. Finally

read the `Employee Onboarding: First Task contract` and reader manifest for this route before completing the named first documentation task with: `python3 tools/validate_content.py --journey journey.employee-onboarding-first-task --semantic-only`. This produces product-side documentation evidence only; it grants no platform authority and performs no approval, publication, promotion, rollback, release-pointer, or live-directory action. Keep missing or conflicting source facts unresolved.

For prospective adopters

Prospective and current customers can use this overview to understand the public-safe value of a governed documentation surface: one semantic source and evidence model is presented consistently across executive, engineering, and AI reading modes. The modes change emphasis, not the underlying claim. This page helps a reader assess whether a bounded documentation use case is worth evaluating; it does not promise a capability, business outcome, performance, compliance result, availability, or customer result. Keep any missing, stale, or conflicting authority visible as an unresolved gap.

Trace the governing claims

Start with `sources/authority.json`, whose approved source list is `source.naming-decision`, `source.documentation-boundary`, and `source.platform-delivery-contract`. In `registries/sources.json`, verify each ID, its recorded locator and revision, and its `public-safe` disclosure: `source.naming-decision` maps to `factory:decisions.md#Governed Execution Documentation Steward naming and route`; `source.documentation-boundary` maps to `product:architecture.md#Semantic source`; and `source.platform-delivery-contract` maps to `agent-orch:docs/delivery-profile-contract.md#content`. The one-semantic-source and evidence-model claim and its reading-mode boundary trace to all three IDs; the no-publication boundary traces to the documentation-boundary and platform-delivery-contract IDs. If an ID, revision, digest, or disclosure is missing or stale, record the claim as unresolved rather than guessing.

Plan your evaluation

Select one bounded documentation use case, write down the claims and evidence it requires, and compare that list with this overview and the three approved registry records. Then request or conduct one scoped evaluation conversation about that comparison. Keep the next step reversible and product-scoped: the conversation evaluates fit and evidence needs, not approval, publication, release-pointer creation, a promised capability, or an adoption result. Record the specific unresolved gaps and stop there if the approved evidence does not establish a conclusion.

Publication Boundaries and Reader Journeys

A reviewed candidate is evidence that a bounded package is ready for review, not evidence of approval or publication. Candidate status remains distinct from promotion, scheduling, rollback, a release pointer, and a live directory. To trace a claim, start at `sources/authority.json`, follow its public-safe IDs through `registries/sources.json`, and use `build/index.json` with the deterministic content-validation, projection, and smoke evidence to reproduce the documented source and evidence trail. For version interpretation, compare the immutable candidate record's previous-version identity, rollback-target identity, rollback availability, and any release-blocking fact; if a fact is missing, stale, or conflicting, record it as unresolved rather than inventing one. Keep product conclusions separate from platform-owned attestations. The next adoption decision is therefore reversible and product-scoped: select one bounded use case, review its evidence gaps, and keep the candidate parked while fit is evaluated. This guidance creates no pointer, promotes nothing, writes no live directory, and does not publish, schedule, or roll back anything.

Candidate-Evidence Preflight Troubleshooting

This is a bounded **Internal Knowledge** troubleshooting route for auto-orch operators and maintainers. Before interpreting any candidate-evidence failure, open `sources/authority.json` and verify `authority_revision: 2026-08-08`, the exact IDs `source.naming-decision`, `source.documentation-boundary`, and `source.platform-delivery-contract`, and `employee_may_modify: false`. Match each ID in `registries/sources.json` and check that its record is disclosed as `public-safe`, with its recorded locator, revision, and digest. Those records map respectively to `factory:decisions.md#Governed Execution Documentation Steward`

`naming and route`, `product:architecture.md#Semantic source`, and `agent-orch:docs/delivery-profile-contract.md#content`. A missing, stale, or conflicting authority or registry fact stays unresolved; it is not repaired by editing prose, fixtures, manifests, or protected sources.

Then use the selected `journeys/candidate_evidence_preflight_troubleshooting.json` as the route declaration for this page. It is mission-authorized, candidate-only, tied to this page, and declares five required, non-exploratory journeys with acceptance traces and exact commands. The contract, model, preflight tool, tests, and fixtures explain the product-side preflight shape; they do not add platform authority. A declaration, expected result, or named review path is not evidence that a command ran or that independent review exists.

Failure Modes

Classify the first observed error before remediation. A schema or evidence-shape failure concerns the required top-level fields `schema_version`, `kind`, `journeys`, and `review_verdict_path`, or a journey's `journey_id`, `name`, `status`, `steps_taken`, `source_refs`, `traces_to`, `blockers`, and `commands_run`. Each command claim also needs `command`, integer `exit_code`, and non-empty `observed_result`; coverage, trace mapping, and a missing observed field are separate failures. A command-allowlist failure means the claimed string is not an exact member of the selected manifest's allowlist. A journey-identity failure means the ID, name, position, or trace does not match the selected declaration. Review-path failures concern the exact `code-reviews/content-review.verdict.json` path and its independently readable artifact. A fixture illustrates a validator case; it is not execution evidence.

For this route, compare commands byte-for-byte against this exact allowlist:

```
python3 -B -m json.tool sources/authority.json
python3 -B -m pytest -p no:cacheprovider tests/test_candidate_evidence_prefli
python3 -B -m json.tool journeys/candidate_evidence_preflight_troubleshooting
python3 -B tools/validate_content.py --semantic-only
```

The selected journey command must also equal that journey's own declaration. Leading or trailing spaces, doubled spaces, newlines, alternate executables, or a command that merely succeeds elsewhere do not

match. An expected result does not replace the actual observed result. Keep the candidate-only boundary and the separation between product conclusions and platform-owned attestations; neither a failure category nor a successful check is transition authority.

Compare journey names byte-for-byte as well. The five selected names are:

```
As an auto-orch operator, verify the sealed authority and public-safe source
As an agent-orch maintainer, diagnose candidate-evidence schema and observed-
As an auto-orch operator, distinguish exact command-allowlist and journey-nam
As a future playbook author, separate fixture examples from independent revie
As an independent reader, verify the troubleshooting guide and route unresolv
```

Do not shorten, punctuate, normalize, or replace a name with an ID. A name or command mismatch remains attached to the original journey and is escalated as declaration drift.

Diagnosis and Remediation

Read `docs/candidate-evidence-preflight-contract.md`, `models/candidate_evidence_handoff_contract.json`, and `tools/candidate_evidence_preflight.py` together with the relevant test and fixture. For schema failures, compare the record with the model and tool: top-level `schema_version: 1`, kind `candidate-evidence-preflight-evidence`, the complete declared journey set, and `review_verdict_path`; then verify each journey's exact name, status, non-empty `steps_taken`, `source_refs`, `traces_to`, and `blockers`. The command claim must preserve the exact declaration, an integer `exit_code`, and a non-empty `observed_result`. The schema permits `passed`, `failed`, `blocked`, and `unresolved`, while the preflight rejects a record whose journey status is not `passed`. Preserve the actual output, status, and blocker; never fill a missing observation with an expected result.

For command drift, compare the complete claimed string byte-for-byte with both the journey declaration and the four-entry allowlist above. The test definitions and `disallowed-command.json` specify a non-allowlisted command as a rejection case, even when it is another plausible content command; that declaration is not proof that a test ran. The same definitions specify leading or trailing spaces, doubled spaces, and newlines as non-matches; there is no normalization step. If a needed check is outside the

allowlist, preserve that mismatch and route a manifest or contract decision to its owner instead of broadening the page or candidate record.

For journey drift, compare `journey_id`, exact `name`, `traces_to`, and array position with the selected manifest. `invalid-journey-name.json` encodes the case where a correct ID cannot rescue a shortened name; it does not record an executed failure. The selected troubleshooting manifest declares five required journeys and its four-command allowlist. The human-readable contract carries a separate set of five canonical names and a two-command allowlist, while the model, executable preflight, and supplied handoff fixtures name `journeys/user_journeys_manifest.json` and the three-journey handoff declarations. These are not interchangeable routes. Recreate evidence only from the current selected declaration and keep any mismatch, path, field, value, expected code, and owner unresolved; do not choose whichever declaration produces a pass.

For the supplied fixture cases, first identify the declaration set they carry. `valid.json` is a complete-shape example for the three-journey handoff shape; it is not automatically valid evidence for this selected five-journey route. `malformed-evidence.json` omits `observed_result`, `disallowed-command.json` uses `python3 tools/validate_content.py`, `invalid-journey-name.json` changes the second exact name, and `missing-review-path.json` omits `review_verdict_path`. These files encode validator cases only; none is execution evidence. The required review artifact is exactly `code-reviews/content-review.verdict.json`; independently verify that it is readable JSON with `schema_version: 1`, a `verdict` of `pass` or `clean`, a `findings` array, and no `High` or `Critical` finding. A named path, fixture shape, or local declaration does not fabricate independent review; preserve a missing, stale, malformed, or unverified verdict as a blocker.

Verification and Escalation

After authority verification, an authorized operator may use only the selected journey's exact allowlisted command. The reproducible record must preserve the repository path, journey ID and byte-exact name, command string, integer exit code, observed output, `steps_taken`, `source_refs`, `traces_to`, `blockers`, and the owner of any unresolved field. A command declaration is not an assertion that it ran, and an expected result is not an observed result. The fixtures are deterministic rejection examples, not execution evidence; the valid fixture is a shape example, not a review result. The exact independently readable review path and its verdict fields must be checked separately.

Escalate when authority or registry data, the selected manifest, contract, model, executable preflight, exact journey name, allowlisted command, observed field, fixture interpretation, or review artifact cannot be reconciled. Include the path, exact field or command, observed value, expected declaration, preserved output, and owner—or explicitly record ownership as unresolved. A failed observation alone does not establish a recurring defect, retry count, affected claim, repair target, route, validator attestation, or gate result. Product-side conclusions remain separate from platform-owned attestations. Keep the handoff parked and candidate-only while the governing owner resolves the declaration or supplies independent evidence. Nothing in this troubleshooting result authorizes approval, activation, promotion, publication, scheduling, rollback, a release pointer, or a live-directory write.

Acceptance Checks

- **AC-1:** Verify authority revision `2026-08-08`, the three exact authority source IDs, and each matching `public-safe` registry record before interpretation; do not infer or modify authority.
- **AC-2:** Diagnose top-level, journey, command-claim, observed-field, status, and coverage failures from the contract, model, tool, tests, and fixtures; preserve actual values and unresolved blockers.
- **AC-3:** Compare each command and each journey name byte-for-byte with the selected manifest and its allowlist; retain the original identity and route declaration drift as unresolved.
- **AC-4:** Distinguish fixture shape examples from execution evidence and independently verify the exact review path and accepted verdict shape; missing, malformed, stale, or unverified review remains a blocker.
- **AC-5:** Record reproducible checks with paths, commands, exit codes, observed output, traces, and ownership, then escalate within the candidate- only boundary while keeping product conclusions separate from platform attestations and every approval or publication transition.

Provenance: registry-listed (no candidate packet)
Registry Source: `content/public/engine-overview.md`