DOCUMENTATION PAGE

Source Authority Orientation

Source Authority Orientation

This Internal Knowledge orientation gives an auto-orch operator a repeatable, read-only path to the authoritative internal source references before interpreting product guidance. It identifies source records and permitted provenance metadata; it does not reproduce source corpora, create authority, or establish a workflow, evaluator, runtime, or transition result.

Authoritative Sources

Begin with `sources/authority.json`. Its sealed revision is `2026-08-08`, its `employee_may_modify` value is `false`, and its ordered `source_files` list is exactly `source.naming-decision`, `source.documentation-boundary`, and `source.platform-delivery-contract`. That file establishes which logical source IDs belong to this orientation. It does not grant permission to edit the authority record or to copy the private material behind a locator.

Reconcile each ID with exactly one record in `registries/sources.json`. The registry exposes only the reference metadata allowed by the contract: authority class, locator, revision, digest, and disclosure. The complete sealed mapping is:

Source ID	Authority class	Locator	Revision	Digest
`source.naming-decision`	`canonical-platform-source`	`factory:decisions.md#Governed Execution Documentation Steward naming and route`	`2026-08-02`	`7c42ccc9f454a81e2c119c14cdc3ad5ba30`
`source.documentation-boundary`	`product-owned-public-safe-source`	`product:architecture.md#Semantic source`	`2026-08-04`	`0094e6f534a04729008a3c85510899e5022`
`source.platform-delivery-contract`	`canonical-platform-source`	`agent-orch:docs/delivery-profile-contract.md#content`	`agent-orch-main-e65a64fcf429`	`d466a6dca8420477f416a535c9c2bd76aa3`

The three records above are the whole source boundary for this page. A convenient Markdown page, `build/index.json`, generated prose, a candidate packet, an evaluator result, a runtime observation, or a command declaration cannot add a source or replace a missing registry record. If an ID, field, revision, digest, or disclosure value is absent, stale, conflicting, or not public-safe, preserve that fact as unresolved and stop source interpretation.

Orientation Workflow

Follow this order for every auto-orch handoff:

1. Read `sources/authority.json` first. Record the authority revision, the `employee_may_modify` flag, and the exact ordered logical source list. Do not add a source because it appears useful or because another page names it.
2. Read `registries/sources.json` and match every listed ID exactly once. Check the authority class, locator, revision, digest, and `public-safe` disclosure from the registered record. Treat these as provenance metadata, not as a summary of the referenced source contents.

3. Use `sources/approved/README.md` for the repository disclosure boundary and keep private governance material, private source corpora, and runtime evidence out of the orientation. A missing or conflicting reference stays visible as an unresolved gap; it is never repaired with plausible prose.
4. Use the synchronized reader journeys in `journeys/source-authority-orientation.json`: follow `journey.source-authority-seal` for the authority boundary, `journey.source-authority-registry-match` for the three registry records, and `journey.source-authority-orientation-boundary` for the read-only interpretation. A command in that manifest is an allowlisted procedure, not proof that the command ran or that an external transition occurred.
5. Before handing off, compare the page metadata with its single registry object and ensure the source references remain the exact sealed set. Keep source identity, product conclusions, candidate evidence, and any platform-owned attestation in their separate evidence boundaries.

This workflow ends with source orientation only. It does not approve, activate, promote, publish, schedule, roll back, change a release pointer, or write a live directory. Those controls are outside this page, and no state about them should be inferred from a synchronized page, generated projection, journey manifest, or successful product-side check.

Source-Identification Check

Use this check before treating an internal reference as authoritative. Confirm that the authority record says revision `2026-08-08`, `employee_may_modify` is `false`, and the source IDs appear in this exact order: `source.naming-decision`, `source.documentation-boundary`, `source.platform-delivery-contract`. Then confirm that the registry contains one and only one record for each ID, with a non-empty authority class, locator, revision, and digest, and with disclosure equal to `public-safe`. Confirm that no extra registry record has been folded into the sealed source list.

The check passes as source orientation only when all of those comparisons resolve and the reader can explain the difference between authority and evidence. `sources/authority.json` supplies source identity; the registry supplies permitted provenance metadata; the contract and journey manifest define how to read that boundary. A page file or `build/index.json` is a generated or semantic product artifact, not an authority source. A candidate packet, evaluator output, runtime observation, or unrun command is evidence or a declaration in its own boundary, not a substitute for the sealed records.

If any comparison fails, write down the exact source ID and field that is missing, stale, conflicting, unregistered, or disallowed, and leave the orientation unresolved. Do not infer a source owner, source contents, defect, repair target, approval, or publication state. The smallest safe next step is to route a permitted follow-up through the owning governed boundary while preserving the original gap. This check supports a traceable operator handoff and nothing more.

Provenance: registry-listed (no candidate packet)
Registry Source: `content/source-authority-orientation.md`