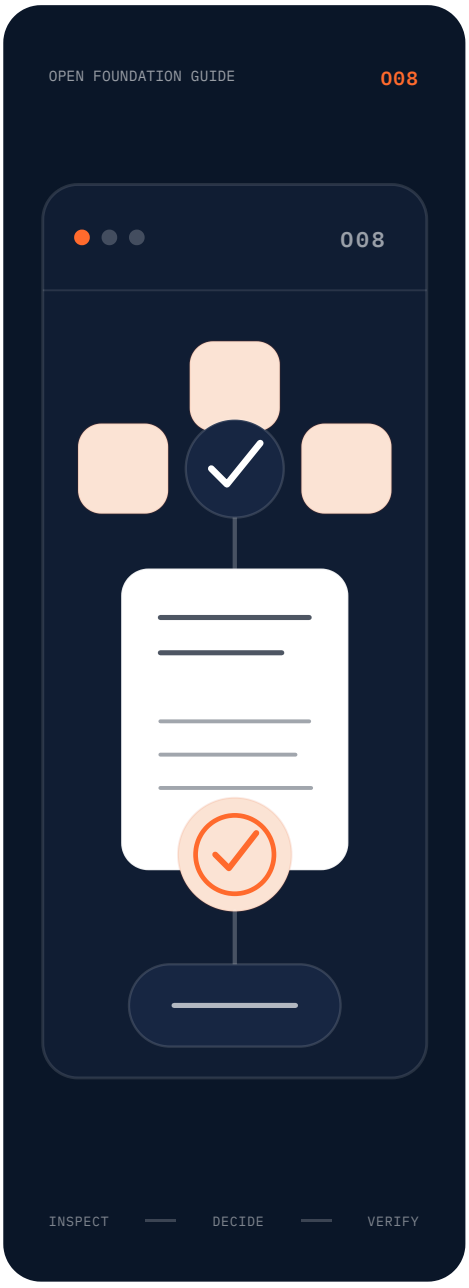
OPEN FOUNDATION GUIDE · 008

Ledger: the complete foundation guide

Mechanism, artefact, interfaces, stack joins, failure, reproducibility, performance method, security, deployment, adoption, and exit for Ledger.

LEDGER MECHANISM ARTEFACT REPRODUCIBILITY SECURITY EXIT



Read this when the name Ledger is not enough and the team needs to inspect the component contract and reproduce its claim.

008

CONTENTS

Ledger, from mechanism to independent check

The guide uses one illustrative exercise: a platform appends typed tool, gate, lifecycle and rights events across a workflow; the event shape and chain remain stable across selected storage backends.

PRIMARY READERS	READING DEPTH	DECISION THIS SUPPORTS
Engineers, maintainers and independent reviewers	Deep technical	A component-level guide for teams deciding whether and how to rely on Ledger for typed, hash-chained system event provenance.

01 **Read this first** 03–04
Scope, reader decision, vocabulary, source and claim status

02 **Mechanism and artefact** 05–12
Responsibility, input, pipeline, manifest, interfaces, stack joins, and failure

03 **Evaluation and adoption** 13–19
Reproducibility, performance, security, deployment, integration, diligence, and exit

04 **Sources and next action** 22
verification records and component proof package

05 **Subject index** 20–21
An alphabetical finder for concepts, controls, products, and decisions.

READING RULE

The published Ledger open-foundation page owns factual statements. Illustrative envelopes and evaluation procedures show what a credible integration should specify; they are not invented project APIs.

SOURCE DISCIPLINE

Dweve-specific facts are taken directly from the English website text audited 2026-07-20; web-navigation wording is humanised for print. Evaluation guidance is editorial, and scenarios and derived visuals are illustrative. Unsupported synthesis is replaced by canonical copy or omitted.

READ THIS FIRST · DECISION

Who should read the Ledger guide

Engineers, architects, security and assurance teams, product owners, open-source reviewers, and operators can use this guide to evaluate typed, hash-chained system event provenance.

Ledger is an inspectable foundation for typed, hash-chained system event provenance. The decision is whether the mechanism and artefact fit a selected product/workflow boundary, can be reproduced on the organisation's inputs and targets, can be packaged and operated safely, and can be replaced without losing data or historical evidence.

The worked example begins when a platform appends typed tool, gate, lifecycle and rights events across a workflow. The expected normal behaviour is: the event shape and chain remain stable across selected storage backends. The fault case is: an event is deleted, reordered or duplicated during migration/replay. A sound result is: integrity or idempotency checks expose the problem without re-executing external side effects. These statements are illustrative evaluation framing, not a customer outcome.

Verify current source/repository and release status, licence, documentation, supported interfaces/platforms, product integration, benchmark method, security, deployment artefacts, support, and rights at the decision date. The website is the sole factual source for this edition; the installed component supplies final proof.

FOUNDATION

Ledger

typed, hash-chained system event provenance

OBJECT

hash-chained typed event stream

The bounded technical artefact.

FAULT

Deliberate negative case

an event is deleted, reordered or duplicated during migration/replay

DECISION

Fit + reproduce

Integrate, operate, verify and exit.

PRIMARY VERIFICATION

Append each selected event category, migrate between two storage tiers without re-encoding, verify chain and schema, alter/delete/reorder events, replay from checkpoint into a read-only projection, test duplicate delivery, and retrieve a rights or incident chronology.

READ THIS FIRST · TERMS

Vocabulary and claim boundary

Foundation, project, surface, artefact, manifest, verifier, benchmark, product integration, and commercial/source rights answer different questions.

Mechanism and artefact

Ledger validates an event's content hash before it appends the event. The event can also carry the previous event's chain hash, so a verifier can detect a missing or reordered entry in the copy it checks. Content-hash recomputation on reads is available through the optional VerifiedStore wrapper.

Project, product, and surface

The open-foundation description describes the project. A product may use it without exposing the same interface or licence. A library, CLI, service, sidecar, FFI, MCP or WASM build is a surface; surface availability and parity require current verification.

Measured and assurance claims

A number remains tied to its corpus/input, hardware, version, configuration, method, baseline and limitation. Integrity, determinism, provenance, containment or offline checking support bounded assurance; they do not prove truth, correctness, legitimacy, legal compliance, fitness or commercial entitlement.

TERM	ANSWERS	DOES NOT ANSWER
Foundation	Which mechanism?	Who owns product work
Artefact	What can be consumed/checked?	Why purpose is legitimate
Surface	How invoked?	Semantic parity automatically
Open/source	What can be inspected?	Current release/right automatically
Benchmark	What was measured?	Production SLA/ROI
Verifier	Which proposition holds?	Every upstream premise is true

Keep these distinctions attached to every adoption claim.

FOUNDATION COUNT WARNING

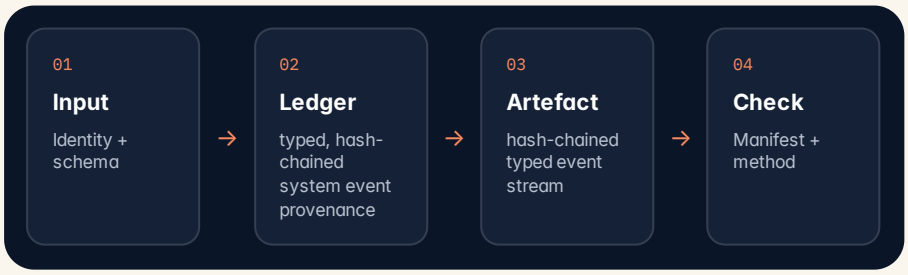
Only relationships asserted by the current product and open-source pages are drawn here. Absence from this ledger is not a claim that a foundation is unused.

01

MECHANISM AND ARTEFACT

Ledger must earn its place at one technical join

Define the problem, input, mechanism, output artefact, interfaces, dependencies, failure, and evidence before discussing adoption.



The component contract used throughout the chapter.

MECHANISM · RESPONSIBILITY

Ledger solves one inspectable technical problem

Ledger is the foundation for typed, hash-chained system event provenance. Its value is the contract and artefact it contributes, not “open source” as a decorative label.

Ledger appends typed events to a hash chain that can be queried and replayed. The same event shape can use memory, JSONL, SQLite, Postgres, or archival storage, and can be embedded through a crate/C ABI or operated through sidecar/service forms.

The worked example begins when a platform appends typed tool, gate, lifecycle and rights events across a workflow. The principal technical object is a hash-chained typed event stream. Under the normal path, the event shape and chain remain stable across selected storage backends. This gives the integration a bounded proposition it can test and retain.

The foundation joins Fabric tenant audit, Nexus organisation history, Aura receipts, Spindle knowledge events, Charter outbox/lifecycle, Knot actions, Selvedge transcripts, and external archival/WORM storage. Each caller should depend on the declared input/output and manifest, not on hidden internal state. A product can validate, route, combine or present the component result while keeping Ledger artefact identity and limitations intact.

FOUNDATION

Ledger

typed, hash-chained system event provenance

OBJECT

hash-chained typed event stream

The artefact or state a caller consumes.

PUBLISHED WEBSITE SOURCES

Ledger open-foundation page

Verify current project/release/source status.

DECISION

Fit + reproduce

Inspect mechanism, integrate, fail, verify, and exit.

PRIMARY BOUNDARY

Ledger shows recorded system events and their integrity. It is not the same as a decision proof or execution transcript, and an event that says “action completed” still needs a trustworthy producer or attached target receipt.

MECHANISM · INPUT CONTRACT

Define the component input before invoking it

A file, query, module, model, event or value becomes a reproducible input only when its identity, schema, configuration, rights, and comparison scope are declared.

For the example, a platform appends typed tool, gate, lifecycle and rights events across a workflow. The component call should bind work/purpose, input identity and origin, schema/format, Ledger version, configuration/profile, target/runtime where relevant, resource and security limits, expected output or verdict, and evidence requirements.

Use content hashes for immutable bytes and signed/versioned identities plus effective metadata for state that cannot be reduced to a file. Preserve collection, parsing, normalisation or translation steps that materially affect meaning. If input is a reference, the manifest must make it resolvable under the selected retention and access policy.

Reject malformed, missing, ambiguous, inaccessible, unauthorised, incompatible, over-limit, or unsupported input before producing a normal-looking output. A valid negative result is different from invalid input; a partial parse or collection is different from empty content; an unsupported target is different from a failed supported execution.

ILLUSTRATIVE LEDGER REQUEST

```
work_id      component-4f9e
purpose      bounded-evaluation
input         content-addressed identity
schema        declared format + version
component    Ledger@pinned
config        profile + target + limits
result        typed artefact or verdict
evidence      manifest + raw check
```

Illustrative envelope, not a claimed current API.

INPUT MUTATION TEST

Change one identity, schema, configuration, target or authority field and verify the result either changes under a new manifest or is rejected explicitly.

MECHANISM · EXECUTION

The Ledger mechanism as a four-stage pipeline

The exact implementation is component-specific; the evaluation pattern remains validate, execute, emit a typed artefact, and bind it to evidence.

Ledger appends typed events to a hash chain that can be queried and replayed. The same event shape can use memory, JSONL, SQLite, Postgres, or archival storage, and can be embedded through a crate/C ABI or operated through sidecar/service forms.

At receive, validate identity, schema, configuration, capability and limits. At execute, run the declared typed, hash-chained system event provenance mechanism without silently substituting an implementation or profile. At emit, return the artefact plus explicit success, negative, partial, refusal, invalid, failure or unverifiable state. At attach, commit versions, inputs, configuration, raw result and verifier dependencies.

The component's described surfaces are Rust crate, C ABI, sidecar, service binary, five documented storage tiers, lifecycle/tool/gate/data-right/incident/attestation events, query and replay. Surface adapters can change transport and ergonomics; they should not change the component proposition. Equivalent supported calls through a library, CLI, service, sidecar, FFI, MCP or WASM surface should reconcile to the same input and artefact identities under the declared mode.

A reusable component-evaluation flow for Ledger.

SURFACE PARITY

Run one supported case through two selected interfaces and compare semantic result, artefact identity, failure class, and evidence, not only transport status.

MECHANISM · RESULT

The artefact manifest is part of the output

The caller needs the Ledger artefact and enough identity to interpret, verify, reproduce, migrate, or reject it later.

The material interface includes stream and event identity, typed schema/version, actor/source, timestamp/order, payload or protected reference, previous hash, event hash, signature/attestation where configured, checkpoint, storage backend, replay cursor, and verification result. The exact object names and schemas come from the current project. The architectural requirement is that the result does not lose the input, component, configuration, target, comparison, and verifier information that gives it meaning.

Return work/result and artefact identity; semantic schema and component versions; input references; configuration/profile; runtime/target; state/error; raw and derived outputs; warnings/limits; integrity or signature; evidence references; and compatibility/retention requirements. Keep human-friendly summary separate from machine-verifiable material.

A downstream product may store only a stable reference when the artefact is sensitive or large. The packet manifest must still resolve the authorised object and make missing, withdrawn, expired or incompatible dependencies explicit. Export should preserve the artefact and manifest, not only a rendered report.

COMPONENT RECORD	ledger-artefact	
COMPONENT	**Ledger + version**	PINNED
INPUT	**identity + schema**	PINNED
CONFIG	**profile + target + limits**	PINNED
RESULT	**hash-chained typed event stream**	TYPED
INTEGRITY	**hash/signature where used**	CHECKED
VERIFIER	**format + runtime + trust**	RETAINED

Illustrative manifest; use the implemented component format.

EXPORT TEST

Export a Ledger result and reproduce the declared check using only the documented artefact, manifest, and verifier dependencies.

MECHANISM · INTERFACES

Integration surfaces must preserve one semantic contract

Ledger can appear as a library, executable, service, sidecar, FFI, MCP or WASM component; each surface is still the same bounded mechanism.

The published description describes Rust crate, C ABI, sidecar, service binary, five documented storage tiers, lifecycle/tool/gate/data-right/incident/attestation events, query and replay. For every selected surface, confirm authentication/authorisation where applicable, schema and version, capability discovery, payload/reference limits, memory/time/fuel or resource limits, streaming/order, concurrency, cancellation, errors, retries, idempotency, logs/evidence, packaging, and supported platform/target.

The application should validate the component result before promoting it into product state. Preserve a stable adapter boundary so an implementation can be upgraded or replaced while the caller's semantic contract remains. Provider-specific data belongs in declared extension fields and the manifest, not in hidden behaviour.

Contract tests drive the same corpus through selected surfaces and compare accepted/rejected input, output state, artefact identity, integrity/verdict, resource behaviour, and failure semantics. Transport codes and process exits should map to documented component result classes.

SURFACE QUESTION	REQUIRED ANSWER
Capability	Which operation/version/profile is supported?
Identity	How are input and artefact named?
Limits	Size, time, memory, concurrency, fuel, target?
Errors	Invalid, negative, partial, failed, unverifiable?
Evidence	Which manifest/raw result/verdict returns?
Packaging	Which runtime/dependencies/rights are required?

The same checklist applies across local and service interfaces.

ADAPTER RULE

An adapter may translate representation; it must not silently weaken the component's correctness, determinism, security, or evidence contract.

MECHANISM · COMPOSITION

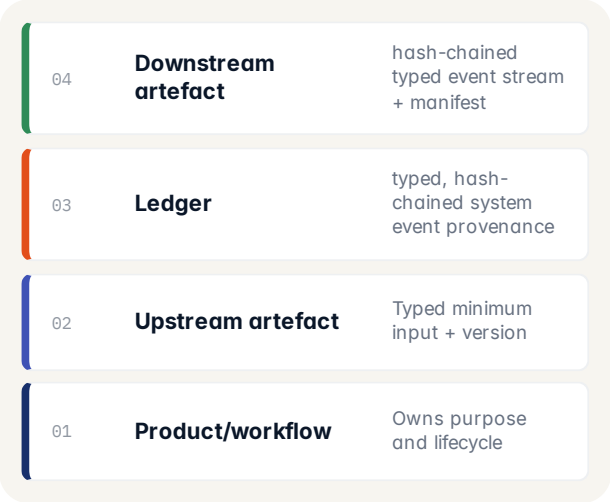
Place the foundation at the correct stack join

A foundation contributes a narrow artefact to products and other foundations; it does not become a product owner merely because it is reusable.

Nexus composes the organisation that performs the work. It is not where your invoices live, not where your contracts are filed, and not the system anybody audits for a balance. Above it, people set the objective and receive the result in Fabric. Beneath it, cognition, governed knowledge and compute are supplied. Your own systems stay yours, reached through recorded crossings.

The upstream caller passes only the minimum input with source, purpose, rights and configuration. The component returns a versioned artefact or reference and state. The downstream caller validates compatibility and may derive new state while preserving lineage. A feedback or update path uses an explicit event/control contract rather than mutating hidden component state.

Mark dependencies required/optional/selected, supported versions, trust/data boundary, resource and determinism effect, packaging and licence, failure mapping, update coupling, and evidence. Then remove or substitute the foundation in a test design to see exactly which promise and object disappear.



A conceptual stack join; exact component dependencies follow the current project manifest.

NO FOUNDATION SOUP

If several foundations appear in a diagram but no one can name the artefact each contributes, the design is branding rather than architecture.

MECHANISM · FAILURE

Failure is part of the component API

The worked fault is: an event is deleted, reordered or duplicated during migration/replay. The safe result is: integrity or idempotency checks expose the problem without re-executing external side effects.

The component-specific failure model is: Schema incompatibility, missing event, order break, hash mismatch, clock anomaly, duplicate append, partial backend migration, corrupt checkpoint, unauthorised payload access, or replay that applies a side effect twice must be detected and contained.

Expose invalid input, unsupported capability/configuration/target, resource or dependency failure, integrity mismatch, verification failure, valid negative/refusal, and partial result as distinct states. Include stable work/input identity, failing stage, version/configuration, retry safety, partial artefact policy, diagnostic detail, and evidence. Never encode all conditions as empty output.

Recovery can correct input, select an explicitly approved substitute, retry a safe pure operation, resume from checkpoint, rebuild a derived artefact, escalate to an owner, quarantine the result, or stop. If a component can cause external effect, reconcile it before retry. Preserve failed attempts and manifest changes.

STATE	MEANING	CALLER ACTION
Invalid	Input/contract cannot be accepted	Correct; do not retry unchanged
Unsupported	Declared capability absent	Select approved path or stop
Negative/refusal	Valid mechanism result	Respect outcome/escalate by policy
Partial	Some material unavailable	Use only if contract permits
Failed	Execution/dependency/resource error	Retry/recover by class
Unverifiable	Integrity/check cannot be established	Quarantine/investigate

Exact state names may differ; the semantic distinctions must remain.

FAULT INJECTION

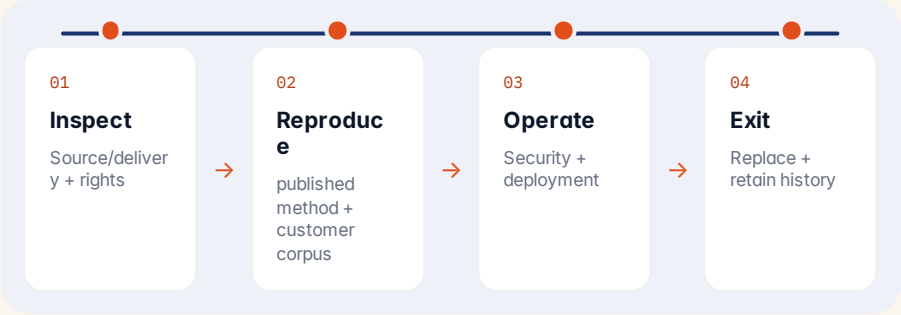
An event is deleted, reordered or duplicated during migration/replay. Verify that integrity or idempotency checks expose the problem without re-executing external side effects.

02

EVALUATION AND ADOPTION

Reproduce, secure, package, integrate, and exit

A foundation becomes production infrastructure only when its narrow claim survives customer inputs, failure, deployment, operations, change, and removal.



The component decision is evidence-backed and reversible.

EVALUATION · REPRODUCIBILITY

Reproduce the Ledger claim, not a summary of it

A credible reproduction pins every identity that can affect the declared proposition and retains raw results, limitations, and negative tests.

The canonical exercise is: Append each selected event category, migrate between two storage tiers without re-encoding, verify chain and schema, alter/delete/reorder events, replay from checkpoint into a read-only projection, test duplicate delivery, and retrieve a rights or incident chronology.

Capture component source or delivered binary identity, build/provenance and runtime environment, dependencies, input/corpus and rights, schemas, configuration/profile, target/hardware/ISA/backend, warm/cold and concurrency conditions where measured, harness/method, repetitions, comparison rule, raw outputs, verdict, reviewer and date.

First reproduce the narrow published claims under its published scope. Then run the customer's representative corpus and target environment. Include malformed, boundary, unsupported, malicious, resource, dependency, integrity and historical cases. A different workload may produce a different performance or quality result without contradicting the original measurement.

COMPONENT RECORD	ledger-reproduction	
COMPONENT	**source/binary + version**	PIN
INPUTS	**corpus + identities + rights**	PIN
ENVIRONMENT	**target + hardware + runtime**	PIN
METHOD	**harness + comparison**	PIN
RESULT	**raw output + verdict**	RETAIN
LIMITS	**scope + exceptions**	RETAIN

A screenshot or supplier summary is not a reproducibility record.

NEGATIVE CONTROL

An event is deleted, reordered or duplicated during migration/replay. The harness should expose the changed condition rather than still returning a green headline.

EVALUATION · PERFORMANCE

Treat every performance number as a measured boundary

Throughput, latency, memory, recall, token reduction, operation coverage, cold start, or energy figures belong to a named workload and method, not the entire platform.

If the Ledger website description publishes a number, preserve its exact component path, corpus or input distribution, size, hardware and instruction set, software version and configuration, warm/cold state, batch/concurrency, repetitions and duration, metric/percentile, accuracy/recall/correctness condition, resource/power method, baseline, exclusions, raw harness and non-SLA qualification.

A fast component can be a small part of end-to-end time; a compact representation can shift work into conversion; an exact result can cost more than an approximate candidate stage; a cold-start advantage can coexist with a slower hot path. Report the full selected path and counter-metrics rather than promoting one route result into universal savings.

Reproduce relied-on numbers as written, then profile the integrated customer workload under steady, burst and degraded conditions. Include preprocessing, transfer, validation, evidence, stores, retries and caller backpressure. Size from service objectives and measured bottlenecks with recovery and growth headroom.

MEASUREMENT FIELD	WHY IT MATTERS
Workload/corpus	Defines what was exercised
Hardware/ISA/backend	Defines execution path
Config/warm/batch	Changes throughput/latency
Correctness/recall	Prevents speed-only comparison
Baseline boundary	Prevents unlike systems comparison
Raw output/repeats	Shows distribution and exceptions

Use this record for every quantitative claim relied on.

NO BENCHMARK ALCHEMY

A component benchmark can inform architecture; it cannot become a production SLA, total-cost saving, or whole-workflow outcome without additional evidence.

EVALUATION · SECURITY

Security includes the component supply chain and parser/runtime boundary

Open or inspectable code helps review; it does not remove build, dependency, input, resource, capability, key, update, and disclosure risks.

Threat-model the Ledger inputs, parsers/decoders or runtime, dependencies, extension/plugin surfaces, FFI or service boundary, resources, files/network where applicable, artefact stores, signatures/keys, verifier, logs, packages, update channel and support/export path. Treat every external input and produced artefact as untrusted until validated.

Request source/repository or delivered binary identity as applicable, licence/rights, build provenance, dependency manifest/SBOM, compiler/toolchain, signatures, vulnerability and disclosure process, unsafe-code posture where stated, sandbox/capability/resource controls where relevant, hardening/fuzz/property tests, and supported platforms/versions.

Test malformed and adversarial inputs, decompression/size/resource exhaustion, path/URL/redirect escape where relevant, parser ambiguity, FFI misuse, service authentication/authorisation, capability denial, modified package, dependency removal, key rotation, verifier abuse, sensitive error/log output, and air-gap closure.

ORIGIN

Provenance

Source/binary, build, dependencies, signature, licence and release status.

INPUT

Robustness

Malformed, malicious, huge, ambiguous, private, unauthorised and incompatible cases.

RUNTIME

Containment

Capabilities, memory/time/fuel, files/network, concurrency and failure isolation.

OUTPUT

Disclosure/integrity

Schema validation, sensitive logs, artefact access, signing, export and verifier limits.

SECURITY BOUNDARY

Ledger shows recorded system events and their integrity. It is not the same as a decision proof or execution transcript, and an event that says “action completed” still needs a trustworthy producer or attached target receipt.

EVALUATION · DEPLOYMENT

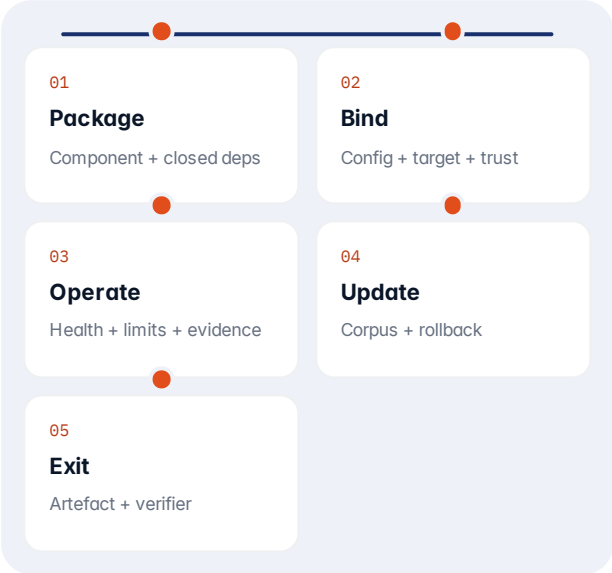
Package and operate Ledger in the selected posture

A foundation may run embedded, as a local executable, sidecar, service, browser/WASM component or another published form; deployment must preserve its contract.

The published surfaces are Rust crate, C ABI, sidecar, service binary, five documented storage tiers, lifecycle/tool/gate/data-right/incident/attestation events, query and replay. Build an installed manifest for component, dependencies, runtime/toolchain where required, configuration/profile, platform/CPU features, stores, models/grammars/rules/keys or other data packages, interfaces/ports, limits, health/telemetry, evidence, licence and documentation.

In managed use, define the service and data boundary. In licensed operation, assign customer owners for package, dependency, access, capacity, observability, updates, backup/restore and support. In air gap, close every runtime, licence, trust, verification, update, and diagnostic dependency and validate controlled-media procedures.

Exercise install/cold start, representative load, dependency/resource failure, update and rollback, backup/restore of retained component state, historical artefact verification, no-network operation where required, supplier disconnect, export and decommission. Preserve exact package and environment versions.



Operational lifecycle for Ledger.

AIR-GAP CHECK

A local executable is not proof of air-gap support if it still requires remote packages, identity, time, keys, licence, telemetry, model, or verification services.

EVALUATION · INTEGRATION

Adopt one reversible component path

Start where the foundation replaces a specific opaque, inconsistent, slow, expensive, unsafe, or unverifiable step, and retain the old path until evidence supports the switch.

Choose one real flow involving Fabric tenant audit, Nexus organisation history, Aura receipts, Spindle knowledge events, Charter outbox/lifecycle, Knot actions, Selvedge transcripts, and external archival/WORM storage. Capture current input/output semantics, quality/correctness, latency/resource/cost, errors, operational work, security boundary, evidence and exit. Define what Ledger changes and what the product/workflow owner continues to own.

Integrate behind a versioned adapter; pin component and dependencies; run side-by-side or shadow where possible; compare semantic results and counter-metrics; inject invalid/failure cases; validate artefact/evidence; measure customer hardware and operations; obtain domain/security/technical approval; and cut over segmentally with rollback.

A trace cannot make an external source, model endpoint, human decision or physical device deterministic. It can identify the dependency, capture the response where policy permits, record the version or attestation available and state how replay treats the boundary. Changed or unavailable dependencies are results, not noise to hide.

01 **Read**
inspect published page, source/delivery, licence, interfaces and tests.

02 **Baseline**
Measure current component boundary and failures.

03 **Integrate**
Versioned adapter and minimum data path.

04 **Compare**
Customer corpus, security, performance and evidence.

05 **Cut over**
Segmented, reversible, monitored and owned.

REVERSIBLE PATH

At each stage the organisation can remove the component and return to the former contract without losing its data, evidence, or ability to explain work.

EVALUATION · DECISION

Due diligence, portability, and the exit test

The foundation is production-ready only when mechanism, availability, rights, reproducibility, security, operation, historical verification, and removal are evidenced for the selected use.

Request the current Ledger source or delivered artefact status, repository/release and documentation, licence/rights, supported surfaces/platforms, semantic schemas/formats, dependency and build manifest, benchmark/verification harnesses, security evidence, known limitations, roadmap status, support/update policy, deployment package and compatibility.

Run the component-specific exercise: Append each selected event category, migrate between two storage tiers without re-encoding, verify chain and schema, alter/delete/reorder events, replay from checkpoint into a read-only projection, test duplicate delivery, and retrieve a rights or incident chronology. Record gaps as verified, not applicable with reason, accepted risk and owner/expiry, contractual condition, remediation and date, or a stop decision. Do not infer “shipping”, “source available”, “MIT”, or product inclusion across components from the open-source overview count.

Binaries, model weights, charts, and every dependency bundle into a signed archive, delivered on encrypted physical media through a documented chain of custody. The import script validates checksums before unpacking. Licence validation is offline-signed. No network egress of any kind after install.

COMPONENT RECORD	ledger-decision	
MECHANISM	**contract + artefact**	INSPECT
AVAILABILITY	**repo/release/delivery**	VERIFY
RIGHTS	**licence/use/modify/support**	CONFIRM
PROOF	**harness + customer corpus**	RUN
OPERATION	**package/update/recover**	PROVE
EXIT	**replace + history verifies**	PROVE

A technical selection should retain this decision and its evidence.

EXIT EXERCISE

Archive the Ledger artefact, manifest, and verifier dependencies; reproduce the declared check without a Dweve-operated service and record what remains dependent.

FIND A SUBJECT

Subject index

Page references point to the substantive explanation, worked example, control, boundary, or decision record, not merely to a passing label.

A **Adoption** 4–5, 13

A **Air-gapped deployment** 17

A **Architecture** 11, 15

A **Artefact identity** 6, 8–10

A **Assurance** 3–4

A **Audit** 6, 18

A **Aura** 6, 18

A **Authority** 7, 10, 16

A **Availability** 4, 19

B **Benchmarking** 3–4, 15, 19

B **Boundaries** 3–4, 6, 10–11, 14–18

C **Capabilities** 8, 10, 12, 16

C **Charter** 6, 18

C **Compliance** 4

C **Configuration** 4, 7–9, 11–12, 14–15, 17

C **Contracts** 5–7, 10–12, 17–19

C **Cost and budgeting** 15, 18

D **Dependencies** 5, 8–12, 14, 16–19

D **Deployment** 3, 13, 17, 19

D **Determinism** 4, 10–11, 18

D **Due diligence** 19

E **Energy** 15

E **Evidence** 3, 5, 7–13, 15, 17–19

E **Exit** 3, 6, 13, 17–19

F **Fabric** 6, 11, 18

F **Failure** 5, 8, 10–13, 16–18

H **Hardware** 4, 14–15, 18

I **Identity** 5–10, 12, 14, 16–17

I **Incident response** 3, 8, 10, 14, 17, 19

I **Inputs** 4–12, 14–16, 18

I **Integration** 3–4, 6, 10, 18

K **Keys and signing** 9, 16

K **Knot** 6, 18

L **Ledger** 3–11, 14–19

L **Licensing** 3–4, 11, 16–19

L **Limits and exclusions** 4, 7–10, 14, 16–17

SUBJECT INDEX CONTINUED

Subject index · L–W

Terms are grouped alphabetically. Consecutive page references are collapsed into ranges.

L **Lineage** 11

M **Manifests** 4–12, 16–17, 19

M **Migration** 3, 12, 14

M **Models and solvers** 7, 12, 16–19

N **Nexus** 6, 11, 18

O **Operations** 10, 12, 15, 17, 19

O **Outputs** 5–7, 9–10, 12, 14–16, 18

O **Ownership** 11–12, 18–19

P **Performance** 14–15, 18

P **Policies** 7, 12, 18–19

P **Portability** 19

P **Proof** 3, 6, 16–17, 19

P **Provenance** 3–6, 8, 11, 14, 16

R **Recovery** 12, 15, 17

R **Replay** 3, 8–10, 12, 14, 17–19

R **Reproducibility** 3, 6, 9, 13–15, 19

R **Responsibility** 6

R **Retention** 7, 9

R **Rights** 3–4, 6–7, 10–11, 13–14, 16, 19

R **Risk** 19

R **Rules** 10, 14

S **Security** 3, 7, 10, 13, 16, 18–19

S **Selvedge** 6, 18

S **Service levels** 4, 15

S **Sources** 3–4, 6, 9, 11, 13–14, 16, 18–19

S **Sovereignty** 3, 6, 8, 12, 16, 18

S **Spindle** 6, 18

S **State** 6–12, 15, 17–18

S **Supply chain** 16

T **Testing** 3, 6–7, 9, 11, 14, 16, 19

T **Trace** 18

V **Validation** 15–16, 19

V **Verification** 3–4, 6–7, 9, 12, 14, 17, 19

V **Versioning** 4, 7, 9–12, 14–15, 18

W **Workflows** 3, 6–7, 11, 15, 18

CLOSING COMPONENT RECORD

The next Ledger action is an independent component proof

Use the real input, configuration, target and failure that matter to the selected workflow. Preserve the artefact and enough dependencies that another authorised team can repeat the check.

Begin with the integration statement: the upstream product/workflow passes a typed input to Ledger for typed, hash-chained system event provenance; the component returns a hash-chained typed event stream and explicit state; the downstream caller validates it and preserves the manifest/evidence. Name every owner and dependency.

Run the primary exercise: Append each selected event category, migrate between two storage tiers without re-encoding, verify chain and schema, alter/delete/reorder events, replay from checkpoint into a read-only projection, test duplicate delivery, and retrieve a rights or incident chronology. Include the negative case, an event is deleted, reordered or duplicated during migration/replay, and confirm that integrity or idempotency checks expose the problem without re-executing external side effects. Capture source/binary, dependencies, inputs/corpus, configuration, target/hardware, method, raw output, verdict, security/failure results and limitations.

Then package and operate it in the intended posture, update and roll back, restore any retained state, verify a historical artefact, export and replace the component, and record what still depends on Dweve. Decide proceed, narrow, remediate, accept with owner/expiry, contract a condition, or stop.

COMPONENT RECORD	ledger-proof	
CONTRACT	**input + output + failure**	INSPECT
ARTEFACT	**identity + manifest**	RETAIN
REPRODUCE	**published method + customer corpus**	RUN
SECURITY	**abuse + supply chain**	RUN
OPERATION	**package/update/recover**	PROVE
EXIT	**replace + history**	PROVE

A component proof ends in an auditable selection decision.

BOUNDARY

Ledger shows recorded system events and their integrity. It is not the same as a decision proof or execution transcript, and an event that says “action completed” still needs a trustworthy producer or attached target receipt.